

GaussDB

Developer Guide(Distributed_V2.0-2.x)

Issue	01
Date	2026-01-06



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2026. 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.....	3
1.4 Concepts.....	4
2 Database Security Management.....	6
2.1 Users and Permissions.....	6
2.1.1 Default Permission Mechanism.....	6
2.1.2 Administrator.....	7
2.1.3 Separation of Duties.....	9
2.1.4 Users.....	11
2.1.5 Roles.....	12
2.1.6 Schemas.....	14
2.1.7 Setting User Permissions.....	15
2.1.8 Row-Level Security Policy.....	16
3 Database Quick Start.....	18
3.1 Connecting to a Database.....	18
3.1.1 Using gsql to Connect to a Database.....	18
3.1.2 APIs.....	19
3.2 Operating a Database.....	20
3.2.1 Creating a Database Account.....	20
3.2.2 Creating and Managing Databases.....	21
3.2.3 Creating and Managing Tablespace.....	23
3.2.4 Creating and Managing Tables.....	26
3.2.4.1 Creating a Table.....	26
3.2.4.2 Inserting Data to a Table.....	26
3.2.4.3 Updating Data in a Table.....	30
3.2.4.4 Viewing Data.....	31
3.2.4.5 Deleting Data from a Table.....	32
3.2.5 Querying a System Catalog.....	32
3.2.6 Other Operations.....	34
3.2.6.1 Creating and Managing Schemas.....	34

3.2.6.2 Creating and Managing Partitioned Tables.....	37
3.2.6.3 Creating and Managing Indexes.....	41
3.2.6.4 Creating and Managing Views.....	45
3.2.6.5 Creating and Managing Sequences.....	46
3.2.6.6 Creating and Managing Scheduled Jobs.....	48
4 Development and Design Proposal.....	52
4.1 Overview.....	52
4.2 Database Design Specifications.....	53
4.2.1 General Specifications.....	53
4.2.2 Deployment Specifications.....	56
4.2.3 Database Object Naming Specifications.....	56
4.2.4 Database and Schema Design Specifications.....	58
4.2.5 Permission Design Specifications.....	59
4.2.6 Character Set Design Specifications.....	60
4.2.7 Table Design Specifications.....	60
4.2.8 Column Design Specifications.....	66
4.2.9 Index Design Specifications.....	69
4.2.10 Function/Stored Procedure Design Specifications.....	70
4.2.11 Constraint Design.....	71
4.2.12 View and Joined Table Design.....	71
4.3 Database Design Specifications.....	72
4.3.1 GUC Parameter Programming Specifications.....	72
4.3.2 Object Access Programming Specifications.....	72
4.3.3 WHERE.....	72
4.3.4 SELECT.....	74
4.3.5 INSERT.....	75
4.3.6 UPDATE.....	76
4.3.7 DELETE.....	76
4.3.8 Join Query.....	77
4.3.9 Subquery.....	77
4.3.10 Transaction.....	78
4.3.11 SQL Compilation.....	79
4.4 Client Programming Specifications.....	83
4.4.1 JDBC.....	83
5 Application Development Guide.....	86
5.1 Development Specifications.....	86
5.2 Development Based on JDBC.....	86
5.2.1 JDBC Package, Driver Class, and Environment Class.....	87
5.2.2 Development Process.....	90
5.2.3 Loading a Driver.....	91
5.2.4 Connecting to a Database.....	91
5.2.5 Connecting to the Database (Using SSL).....	105

5.2.6 Connecting to a Database (Using UDS).....	106
5.2.7 Running SQL Statements.....	107
5.2.8 Processing Data in a Result Set.....	112
5.2.9 Closing a Database Connection.....	114
5.2.10 Log Management.....	115
5.2.11 Example: Common Operations.....	118
5.2.12 Example: Retrying SQL Queries for Applications.....	122
5.2.13 Example: Importing and Exporting Data Through Local Files.....	126
5.2.14 Example: Migrating Data from MySQL.....	128
5.2.15 Example: Logic Replication Code.....	129
5.2.16 Example: Parameters for Connecting to the Database in Different Scenarios.....	136
5.2.17 JDBC API Reference.....	137
5.2.17.1 java.sql.Connection.....	137
5.2.17.2 java.sql.CallableStatement.....	140
5.2.17.3 java.sql.DatabaseMetaData.....	142
5.2.17.4 java.sql.Driver.....	150
5.2.17.5 java.sql.PreparedStatement.....	151
5.2.17.6 java.sql.ResultSet.....	154
5.2.17.7 java.sql.ResultSetMetaData.....	161
5.2.17.8 java.sql.Statement.....	162
5.2.17.9 javax.sql.ConnectionPoolDataSource.....	165
5.2.17.10 javax.sql.DataSource.....	165
5.2.17.11 javax.sql.PooledConnection.....	166
5.2.17.12 javax.naming.Context.....	166
5.2.17.13 javax.naming.spi.InitialContextFactory.....	167
5.2.17.14 CopyManager.....	167
5.2.17.15 PGReplicationConnection.....	169
5.2.17.16 PGReplicationStream.....	169
5.2.17.17 ChainedStreamBuilder.....	171
5.2.17.18 ChainedCommonStreamBuilder.....	171
5.2.18 Mapping for JDBC Data Types.....	172
5.2.19 Troubleshooting.....	174
5.2.19.1 Incorrect batchSize Settings.....	174
5.2.19.2 Error Is Reported or Connection Is Blocked in SSL Mode.....	175
5.3 Development Based on ODBC.....	176
5.3.1 ODBC Packages, Dependent Libraries, and Header Files.....	177
5.3.2 Development Process.....	179
5.3.3 Configuring a Data Source in the Linux OS.....	181
5.3.4 Configuring a Data Source in the Windows OS.....	192
5.3.5 Example: Common Functions and Batch Binding.....	196
5.3.6 Typical Application Scenarios and Configurations.....	203
5.3.7 ODBC Interface Reference.....	211

5.3.7.1 SQLAllocEnv.....	211
5.3.7.2 SQLAllocConnect.....	211
5.3.7.3 SQLAllocHandle.....	212
5.3.7.4 SQLAllocStmt.....	213
5.3.7.5 SQLBindCol.....	213
5.3.7.6 SQLBindParameter.....	214
5.3.7.7 SQLColAttribute.....	216
5.3.7.8 SQLConnect.....	217
5.3.7.9 SQLDisconnect.....	218
5.3.7.10 SQLExecDirect.....	219
5.3.7.11 SQLExecute.....	220
5.3.7.12 SQLFetch.....	221
5.3.7.13 SQLFreeStmt.....	222
5.3.7.14 SQLFreeConnect.....	222
5.3.7.15 SQLFreeHandle.....	222
5.3.7.16 SQLFreeEnv.....	223
5.3.7.17 SQLPrepare.....	223
5.3.7.18 SQLGetData.....	224
5.3.7.19 SQLGetDiagRec.....	226
5.3.7.20 SQLSetConnectAttr.....	228
5.3.7.21 SQLSetEnvAttr.....	229
5.3.7.22 SQLSetStmtAttr.....	230
5.4 Development Based on libpq.....	232
5.4.1 libpq Package, Dependency Library, and Header File.....	232
5.4.2 Development Process.....	233
5.4.3 Example.....	233
5.4.4 libpq API Reference.....	239
5.4.4.1 Database Connection Control Functions.....	239
5.4.4.1.1 PQconnectdbParams.....	239
5.4.4.1.2 PQconnectdb.....	240
5.4.4.1.3 PQbackendPID.....	241
5.4.4.1.4 PQsetdbLogin.....	241
5.4.4.1.5 PQfinish.....	242
5.4.4.1.6 PQreset.....	243
5.4.4.1.7 PQstatus.....	243
5.4.4.2 Database Statement Execution Functions.....	244
5.4.4.2.1 PQexec.....	245
5.4.4.2.2 PQprepare.....	246
5.4.4.2.3 PQresultStatus.....	247
5.4.4.2.4 PQclear.....	248
5.4.4.3 Functions for Asynchronous Command Processing.....	249
5.4.4.3.1 PQsendQuery.....	249

5.4.4.3.2 PQsendQueryParams.....	250
5.4.4.3.3 PQsendPrepare.....	251
5.4.4.3.4 PQsendQueryPrepared.....	252
5.4.4.3.5 PQflush.....	253
5.4.4.4 Functions for Canceling Queries in Progress.....	254
5.4.4.4.1 PQgetCancel.....	254
5.4.4.4.2 PQfreeCancel.....	254
5.4.4.4.3 PQcancel.....	255
5.4.5 Connection Parameters.....	256
5.5 Psycopg-Based Development.....	260
5.5.1 Development Process.....	261
5.5.2 Psycopg Package.....	262
5.5.3 Example: Common Operations.....	266
5.5.4 Psycopg API Reference.....	267
5.5.4.1 psycopg2.connect().....	267
5.5.4.2 connection.cursor().....	269
5.5.4.3 cursor.execute(query,vars_list).....	270
5.5.4.4 cursor.executemany(query,vars_list).....	270
5.5.4.5 connection.commit().....	271
5.5.4.6 connection.rollback().....	271
5.5.4.7 cursor.fetchone().....	272
5.5.4.8 cursor.fetchall().....	272
5.5.4.9 cursor.close().....	273
5.5.4.10 connection.close().....	273
5.6 Commissioning.....	274
6 SQL Optimization.....	279
6.1 Query Execution Process.....	279
6.2 Introduction to the SQL Execution Plan.....	282
6.2.1 Overview.....	282
6.2.2 Description.....	283
6.3 Optimization Process.....	292
6.4 Updating Statistics.....	293
6.5 Reviewing and Modifying a Table Definition.....	293
6.5.1 Overview.....	293
6.5.2 Selecting a Storage Model.....	294
6.5.3 Selecting a Distribution Mode.....	294
6.5.4 Selecting Distribution Keys.....	296
6.5.5 Using Partitioned Tables.....	296
6.5.6 Selecting a Data Type.....	297
6.6 Typical SQL Optimization Methods.....	297
6.6.1 Optimizing Statement Pushdown.....	297
6.6.2 Optimizing Subqueries.....	306

6.6.3 Optimizing Statistics.....	314
6.6.4 Optimizing Operators.....	320
6.6.5 Optimizing Data Skew.....	321
6.7 Experience in Rewriting SQL Statements.....	327
6.8 Configuring Key Parameters for SQL Optimization.....	329
6.9 Hint-based Tuning.....	331
6.9.1 Plan Hint Optimization.....	331
6.9.2 Join Order Hints.....	334
6.9.3 Join Operation Hints.....	336
6.9.4 Rows Hints.....	336
6.9.5 Stream Operation Hints.....	338
6.9.6 Scan Operation Hints.....	340
6.9.7 Sublink Name Hints.....	341
6.9.8 Skew Hints.....	342
6.9.9 Parameterized Path Hint.....	346
6.9.10 Hint Errors, Conflicts, and Other Warnings.....	347
6.9.11 Plan Hint Cases.....	349
6.9.12 Optimizer GUC Parameter Hints.....	353
6.9.13 Hints for Selecting the Custom Plan or Generic Plan.....	355
6.9.14 Hints Specifying Not to Expand Subqueries.....	356
6.9.15 Hints Specifying Not to Use Global Plan Cache.....	357
6.9.16 Hint of Parameterized Paths at the Same Level.....	357
6.10 Checking the Implicit Conversion Performance.....	359
6.11 Optimization Cases.....	360
6.11.1 Case: Selecting an Appropriate Distribution Key.....	360
6.11.2 Case: Creating an Appropriate Index.....	361
6.11.3 Case: Adding NOT NULL for JOIN Columns.....	362
6.11.4 Case: Pushing Down Sort Operations to DNs.....	364
6.11.5 Case: Setting cost_param and Optimizing Query Performance.....	365
6.11.6 Case: Adjusting Distribution Keys.....	369
6.11.7 Case: Adjusting Partial Clustering Keys.....	370
6.11.8 Case: Modifying a Partitioned Table.....	371
6.11.9 Case: Adjusting the GUC Parameter best_agg_plan.....	371
6.11.10 Case: Rewriting SQL Statements to Eliminate Subqueries.....	373
6.11.11 Case: Rewriting SQL Statements to Eliminate Pruning Interference.....	373
6.11.12 Case: Rewriting SQL Statements and Deleting in-clause.....	374
6.11.13 Case: Modifying the GUC Parameter rewrite_rule.....	376
7 SQL Reference.....	383
7.1 GaussDB SQL.....	383
7.2 Keywords.....	384
7.3 Data Type.....	419
7.3.1 Numeric Types.....	420

7.3.2 Monetary Types.....	426
7.3.3 Boolean Types.....	427
7.3.4 Character Types.....	428
7.3.5 Binary Types.....	431
7.3.6 Date/Time Types.....	432
7.3.7 Geometric Types.....	442
7.3.8 Network Address Types.....	444
7.3.9 Bit String Types.....	448
7.3.10 UUID Type.....	449
7.3.11 JSON/JSONB Types.....	449
7.3.12 HLL.....	453
7.3.13 Range Types.....	457
7.3.14 Object Identifier Types.....	461
7.3.15 Pseudo-Types.....	463
7.3.16 ACLItem.....	465
7.4 Constant and Macro.....	465
7.5 Functions and Operators.....	466
7.5.1 Logical Operators.....	467
7.5.2 Comparison Operators.....	467
7.5.3 Character Processing Functions and Operators.....	468
7.5.4 Binary String Functions and Operators.....	492
7.5.5 Bit String Functions and Operators.....	496
7.5.6 Pattern Matching Operators.....	498
7.5.7 Mathematical Functions and Operators.....	502
7.5.8 Date and Time Processing Functions and Operators.....	517
7.5.9 Type Conversion Functions.....	540
7.5.10 Geometric Functions and Operators.....	557
7.5.11 Network Address Functions and Operators.....	568
7.5.12 JSON/JSONB Functions and Operators.....	573
7.5.13 HLL Functions and Operators.....	585
7.5.14 SEQUENCE Functions.....	598
7.5.15 Array Functions and Operators.....	601
7.5.16 Range Functions and Operators.....	608
7.5.17 Aggregate Functions.....	613
7.5.18 Window Functions.....	626
7.5.19 Security Functions.....	631
7.5.20 Set Returning Functions.....	637
7.5.21 Conditional Expression Functions.....	639
7.5.22 System Information Functions.....	641
7.5.23 System Administration Functions.....	679
7.5.23.1 Configuration Settings Functions.....	679
7.5.23.2 Universal File Access Functions.....	680

7.5.23.3 Server Signal Functions.....	682
7.5.23.4 Backup and Restoration Control Functions.....	684
7.5.23.5 Dual-Cluster DR Control Functions.....	690
7.5.23.6 Dual-Cluster DR Query Functions.....	692
7.5.23.7 Snapshot Synchronization Functions.....	696
7.5.23.8 Database Object Functions.....	697
7.5.23.9 Advisory Lock Functions.....	702
7.5.23.10 Logical Replication Functions.....	705
7.5.23.11 Segment-Page Storage Functions.....	716
7.5.23.12 Other Functions.....	719
7.5.24 Statistics Information Functions.....	744
7.5.25 Trigger Functions.....	803
7.5.26 Hash Function.....	804
7.5.27 Prompt Message Function.....	811
7.5.28 Fault Injection System Function.....	812
7.5.29 Redistribution Parameters.....	812
7.5.30 Distribution Key Recommendation Functions.....	812
7.5.31 Internal Functions.....	818
7.5.32 Dynamic Data Masking Functions.....	823
7.5.33 Hotkey Feature Functions.....	824
7.5.34 Global SysCache Feature Functions.....	824
7.5.35 Data Damage Detection and Repair Functions.....	827
7.5.36 Other System Functions.....	832
7.5.37 Obsolete Functions.....	857
7.6 Expressions.....	858
7.6.1 Simple Expressions.....	858
7.6.2 Condition Expressions.....	859
7.6.3 Subquery Expressions.....	863
7.6.4 Array Expressions.....	866
7.6.5 Row Expressions.....	868
7.7 Type Conversion.....	869
7.7.1 Overview.....	869
7.7.2 Operators.....	871
7.7.3 Functions.....	873
7.7.4 Value Storage.....	875
7.7.5 UNION, CASE, and Related Constructs.....	876
7.8 System Operation.....	880
7.9 Controlling Transactions.....	881
7.10 DDL Syntax Overview.....	882
7.11 DML Syntax Overview.....	885
7.12 DCL Syntax Overview.....	886
7.13 SQL Syntax.....	887

7.13.1 ABORT.....	887
7.13.2 ALTER AUDIT POLICY.....	889
7.13.3 ALTER COORDINATOR.....	890
7.13.4 ALTER DATABASE.....	891
7.13.5 ALTER DEFAULT PRIVILEGES.....	894
7.13.6 ALTER DIRECTORY.....	896
7.13.7 ALTER FUNCTION.....	897
7.13.8 ALTER GLOBAL CONFIGURATION.....	900
7.13.9 ALTER GROUP.....	901
7.13.10 ALTER INDEX.....	902
7.13.11 ALTER LANGUAGE.....	904
7.13.12 ALTER MASKING POLICY.....	904
7.13.13 ALTER MATERIALIZED VIEW.....	906
7.13.14 ALTER NODE.....	908
7.13.15 ALTER NODE GROUP.....	908
7.13.16 ALTER RESOURCE LABEL.....	910
7.13.17 ALTER ROLE.....	911
7.13.18 ALTER ROW LEVEL SECURITY POLICY.....	913
7.13.19 ALTER SCHEMA.....	915
7.13.20 ALTER SEQUENCE.....	916
7.13.21 ALTER SERVER.....	918
7.13.22 ALTER SESSION.....	919
7.13.23 ALTER SYNONYM.....	921
7.13.24 ALTER SYSTEM KILL SESSION.....	922
7.13.25 ALTER TABLE.....	923
7.13.26 ALTER TABLE PARTITION.....	938
7.13.27 ALTER TABLESPACE.....	942
7.13.28 ALTER TRIGGER.....	944
7.13.29 ALTER TYPE.....	945
7.13.30 ALTER USER.....	947
7.13.31 ALTER VIEW.....	949
7.13.32 ANALYZE ANALYSE.....	951
7.13.33 BEGIN.....	954
7.13.34 CALL.....	955
7.13.35 CHECKPOINT.....	957
7.13.36 CLEAN CONNECTION.....	958
7.13.37 CLOSE.....	959
7.13.38 CLUSTER.....	960
7.13.39 COMMENT.....	962
7.13.40 COMMIT END.....	965
7.13.41 COMMIT PREPARED.....	966
7.13.42 COPY.....	967

7.13.43 CREATE AUDIT POLICY.....	982
7.13.44 CREATE BARRIER.....	984
7.13.45 CREATE CONVERSION.....	984
7.13.46 CREATE DATABASE.....	985
7.13.47 CREATE DIRECTORY.....	993
7.13.48 CREATE FUNCTION.....	995
7.13.49 CREATE GROUP.....	1002
7.13.50 CREATE INCREMENTAL MATERIALIZED VIEW.....	1003
7.13.51 CREATE INDEX.....	1005
7.13.52 CREATE LANGUAGE.....	1012
7.13.53 CREATE MASKING POLICY.....	1012
7.13.54 CREATE MATERIALIZED VIEW.....	1014
7.13.55 CREATE NODE.....	1016
7.13.56 CREATE NODE GROUP.....	1017
7.13.57 CREATE PROCEDURE.....	1018
7.13.58 CREATE RESOURCE LABEL.....	1022
7.13.59 CREATE ROLE.....	1023
7.13.60 CREATE ROW LEVEL SECURITY POLICY.....	1028
7.13.61 CREATE SCHEMA.....	1032
7.13.62 CREATE SEQUENCE.....	1033
7.13.63 CREATE SERVER.....	1036
7.13.64 CREATE SYNONYM.....	1038
7.13.65 CREATE TABLE.....	1040
7.13.66 CREATE TABLESPACE.....	1061
7.13.67 CREATE TABLE AS.....	1063
7.13.68 CREATE TABLE PARTITION.....	1067
7.13.69 CREATE TRIGGER.....	1081
7.13.70 CREATE TYPE.....	1087
7.13.71 CREATE USER.....	1094
7.13.72 CREATE VIEW.....	1096
7.13.73 CREATE WEAK PASSWORD DICTIONARY.....	1097
7.13.74 CURSOR.....	1098
7.13.75 DEALLOCATE.....	1100
7.13.76 DECLARE.....	1100
7.13.77 DELETE.....	1102
7.13.78 DO.....	1104
7.13.79 DROP AUDIT POLICY.....	1105
7.13.80 DROP DATABASE.....	1106
7.13.81 DROP DIRECTORY.....	1107
7.13.82 DROP FUNCTION.....	1108
7.13.83 DROP GLOBAL CONFIGURATION.....	1109
7.13.84 DROP GROUP.....	1109

7.13.85 DROP INDEX.....	1110
7.13.86 DROP LANGUAGE.....	1110
7.13.87 DROP MASKING POLICY.....	1111
7.13.88 DROP MATERIALIZED VIEW.....	1111
7.13.89 DROP NODE.....	1112
7.13.90 DROP NODE GROUP.....	1113
7.13.91 DROP OWNED.....	1113
7.13.92 DROP PROCEDURE.....	1114
7.13.93 DROP RESOURCE LABEL.....	1115
7.13.94 DROP ROLE.....	1115
7.13.95 DROP ROW LEVEL SECURITY POLICY.....	1116
7.13.96 DROP SCHEMA.....	1117
7.13.97 DROP SEQUENCE.....	1118
7.13.98 DROP SERVER.....	1118
7.13.99 DROP SYNONYM.....	1119
7.13.100 DROP TABLE.....	1120
7.13.101 DROP TABLESPACE.....	1121
7.13.102 DROP TRIGGER.....	1122
7.13.103 DROP TYPE.....	1122
7.13.104 DROP USER.....	1123
7.13.105 DROP VIEW.....	1125
7.13.106 DROP WEAK PASSWORD DICTIONARY.....	1125
7.13.107 EXECUTE.....	1126
7.13.108 EXECUTE DIRECT.....	1127
7.13.109 EXPLAIN.....	1128
7.13.110 EXPLAIN PLAN.....	1133
7.13.111 FETCH.....	1135
7.13.112 GRANT.....	1139
7.13.113 INSERT.....	1148
7.13.114 LOCK.....	1153
7.13.115 MOVE.....	1157
7.13.116 MERGE INTO.....	1158
7.13.117 PREPARE.....	1161
7.13.118 PREPARE TRANSACTION.....	1161
7.13.119 REASSIGN OWNED.....	1162
7.13.120 REINDEX.....	1163
7.13.121 REFRESH INCREMENTAL MATERIALIZED VIEW.....	1165
7.13.122 REFRESH MATERIALIZED VIEW.....	1166
7.13.123 RELEASE SAVEPOINT.....	1167
7.13.124 RESET.....	1168
7.13.125 REVOKE.....	1169
7.13.126 ROLLBACK.....	1173

7.13.127 ROLLBACK PREPARED.....	1174
7.13.128 ROLLBACK TO SAVEPOINT.....	1174
7.13.129 SAVEPOINT.....	1175
7.13.130 SELECT.....	1177
7.13.131 SELECT INTO.....	1190
7.13.132 SET.....	1192
7.13.133 SET CONSTRAINTS.....	1194
7.13.134 SET ROLE.....	1195
7.13.135 SET SESSION AUTHORIZATION.....	1196
7.13.136 SET TRANSACTION.....	1197
7.13.137 SHOW.....	1198
7.13.138 SHUTDOWN.....	1199
7.13.139 START TRANSACTION.....	1200
7.13.140 TRUNCATE.....	1201
7.13.141 UPDATE.....	1204
7.13.142 VACUUM.....	1207
7.13.143 VALUES.....	1210
7.14 Appendix.....	1211
7.14.1 Extended Functions.....	1211
8 Best Practices.....	1212
8.1 Best Practices of Table Design.....	1212
8.1.1 Selecting a Storage Model.....	1212
8.1.2 Selecting a Distribution Mode.....	1212
8.1.3 Selecting Distribution Keys.....	1213
8.1.4 Using Partitioned Tables.....	1214
8.1.5 Selecting a Data Type.....	1214
8.1.6 Checking a Node Where a Table Resides.....	1215
8.2 Best Practices of SQL Queries.....	1215
8.3 Best Practices for Data Skew Query.....	1217
8.3.1 Quickly Locating Tables That Cause Data Skew.....	1217
9 User-defined Functions.....	1219
9.1 PL/SQL Functions.....	1219
10 Stored Procedures.....	1220
10.1 Overview.....	1220
10.2 Data Types.....	1220
10.3 Data Type Conversion.....	1220
10.4 Arrays and Records.....	1221
10.4.1 Arrays.....	1222
10.4.2 Records.....	1223
10.5 DECLARE Syntax.....	1225
10.5.1 Basic Structure.....	1225

10.5.2 Anonymous Blocks.....	1226
10.5.3 Subprograms.....	1227
10.6 Basic Statements.....	1227
10.6.1 Variable Definition Statements.....	1227
10.6.2 Assignment Statements.....	1229
10.6.3 Call Statements.....	1230
10.7 Dynamic Statements.....	1231
10.7.1 Executing Dynamic Query Statements.....	1231
10.7.2 Executing Dynamic Non-Query Statements.....	1234
10.7.3 Dynamically Calling Stored Procedures.....	1236
10.7.4 Dynamically Calling Anonymous Blocks.....	1237
10.8 Control Statements.....	1239
10.8.1 RETURN Statements.....	1239
10.8.1.1 RETURN.....	1239
10.8.1.2 RETURN NEXT and RETURN QUERY.....	1239
10.8.2 Conditional Statements.....	1240
10.8.3 Loop Statements.....	1243
10.8.4 Branch Statements.....	1246
10.8.5 NULL Statements.....	1247
10.8.6 Error Trapping Statements.....	1247
10.8.7 GOTO Statements.....	1250
10.9 Transaction Statements.....	1251
10.10 Other Statements.....	1259
10.10.1 Lock Operations.....	1259
10.10.2 Cursor Operations.....	1259
10.11 Cursors.....	1260
10.11.1 Overview.....	1260
10.11.2 Explicit Cursor.....	1260
10.11.3 Implicit Cursor.....	1265
10.11.4 Cursor Loop.....	1266
10.12 Advanced Packages.....	1267
10.12.1 Basic APIs.....	1267
10.12.1.1 PKG_SERVICE.....	1267
10.12.1.2 PKG_UTIL.....	1278
10.12.2 Secondary Encapsulation APIs (Recommended).....	1299
10.12.2.1 DBE_LOB.....	1299
10.12.2.2 DBE_RANDOM.....	1312
10.12.2.3 DBE_OUTPUT.....	1314
10.12.2.4 DBE_RAW.....	1315
10.12.2.5 DBE_TASK.....	1319
10.12.2.6 DBE_UTILITY.....	1330
10.12.2.7 DBE_SQL.....	1331

10.12.2.8 DBE_FILE.....	1358
10.12.2.9 DBE_SESSION.....	1379
10.12.2.10 DBE_MATCH.....	1381
10.12.2.11 DBE_SCHEDULER.....	1382
10.12.2.12 DBE_APPLICATION_INFO.....	1421
10.13 Retry Management.....	1422
10.14 Debugging.....	1423
11 Autonomous Transaction.....	1426
11.1 Stored Procedure Supporting Autonomous Transaction.....	1426
11.2 Anonymous Block Supporting Autonomous Transaction.....	1427
11.3 Function Supporting Autonomous Transaction.....	1427
11.4 Restrictions.....	1428
12 System Catalogs and System Views.....	1431
12.1 Overview of System Catalogs and System Views.....	1431
12.2 System Catalogs.....	1431
12.2.1 GS_AUDITING_POLICY.....	1432
12.2.2 GS_AUDITING_POLICY_ACCESS.....	1432
12.2.3 GS_AUDITING_POLICY_FILTERS.....	1433
12.2.4 GS_AUDITING_POLICY_PRIVILEGES.....	1433
12.2.5 GS_ASP.....	1434
12.2.6 GS_DB_PRIVILEGE.....	1436
12.2.7 GS_GLOBAL_CONFIG.....	1436
12.2.8 GS_JOB_ATTRIBUTE.....	1437
12.2.9 GS_JOB_ARGUMENT.....	1437
12.2.10 GS_MASKING_POLICY.....	1438
12.2.11 GS_MASKING_POLICY_ACTIONS.....	1439
12.2.12 GS_MASKING_POLICY_FILTERS.....	1439
12.2.13 GS_MATVIEW.....	1440
12.2.14 GS_MATVIEW_DEPENDENCY.....	1440
12.2.15 GS_OPT_MODEL.....	1441
12.2.16 GS_POLICY_LABEL.....	1441
12.2.17 GS_RECYCLEBIN.....	1442
12.2.18 GS_SQL_PATCH.....	1442
12.2.19 GS_TXN_SNAPSHOT.....	1443
12.2.20 GS_UID.....	1443
12.2.21 PG_AGGREGATE.....	1443
12.2.22 PG_AM.....	1445
12.2.23 PG_AMOP.....	1448
12.2.24 PG_AMPROC.....	1449
12.2.25 PG_APP_WORKLOADGROUP_MAPPING.....	1450
12.2.26 PG_ATTRDEF.....	1450
12.2.27 PG_ATTRIBUTE.....	1451

12.2.28 PG_AUTHID.....	1452
12.2.29 PG_AUTH_HISTORY.....	1455
12.2.30 PG_AUTH_MEMBERS.....	1455
12.2.31 PG_CAST.....	1456
12.2.32 PG_CLASS.....	1456
12.2.33 PG_COLLATION.....	1461
12.2.34 PG_CONSTRAINT.....	1461
12.2.35 PG_CONVERSION.....	1464
12.2.36 PG_DATABASE.....	1465
12.2.37 PG_DB_ROLE_SETTING.....	1466
12.2.38 PG_DEFAULT_ACL.....	1467
12.2.39 PG_DEPEND.....	1467
12.2.40 PG_DESCRIPTION.....	1469
12.2.41 PG_DIRECTORY.....	1469
12.2.42 PG_ENUM.....	1470
12.2.43 PG_FOREIGN_SERVER.....	1470
12.2.44 PG_HASHBUCKET.....	1471
12.2.45 PG_INDEX.....	1471
12.2.46 PG_INHERITS.....	1474
12.2.47 PG_JOB.....	1474
12.2.48 PG_JOB_PROC.....	1476
12.2.49 PG_LANGUAGE.....	1477
12.2.50 PG_LARGEOBJECT.....	1478
12.2.51 PG_LARGEOBJECT_METADATA.....	1478
12.2.52 PG_NAMESPACE.....	1479
12.2.53 PG_OBJECT.....	1479
12.2.54 PG_OPCLASS.....	1481
12.2.55 PG_OPERATOR.....	1481
12.2.56 PG_OPFAMILY.....	1482
12.2.57 PG_PARTITION.....	1483
12.2.58 PG_PLTEMPLATE.....	1485
12.2.59 PG_PROC.....	1486
12.2.60 PG_RANGE.....	1490
12.2.61 PG_REPLICATION_ORIGIN.....	1491
12.2.62 PG_RESOURCE_POOL.....	1491
12.2.63 PG_REWRITE.....	1492
12.2.64 PG_RLSPOLICY.....	1493
12.2.65 PG_SECLABEL.....	1494
12.2.66 PG_SHDEPEND.....	1494
12.2.67 PG_SHDESCRIPTION.....	1496
12.2.68 PG_SHSECLABEL.....	1496
12.2.69 PG_STATISTIC.....	1497

12.2.70 PG_STATISTIC_EXT.....	1498
12.2.71 PG_SYNONYM.....	1500
12.2.72 PG_TABLESPACE.....	1500
12.2.73 PG_TRIGGER.....	1501
12.2.74 PG_TS_CONFIG.....	1502
12.2.75 PG_TS_CONFIG_MAP.....	1503
12.2.76 PG_TS_DICT.....	1503
12.2.77 PG_TS_PARSER.....	1504
12.2.78 PG_TS_TEMPLATE.....	1505
12.2.79 PG_TYPE.....	1505
12.2.80 PG_USER_MAPPING.....	1509
12.2.81 PG_USER_STATUS.....	1509
12.2.82 PG_WORKLOAD_GROUP.....	1510
12.2.83 PGXC_CLASS.....	1511
12.2.84 PGXC_GROUP.....	1511
12.2.85 PGXC_NODE.....	1512
12.2.86 PGXC_REDISTB.....	1514
12.2.87 PGXC_SLICE.....	1515
12.2.88 PLAN_TABLE_DATA.....	1516
12.2.89 STATEMENT_HISTORY.....	1518
12.2.90 STREAMING_STREAM.....	1523
12.2.91 STREAMING_CONT_QUERY.....	1523
12.2.92 STREAMING_REAPER_STATUS.....	1524
12.3 System Views.....	1524
12.3.1 ADM_COL_COMMENTS.....	1524
12.3.2 ADM_CONS_COLUMNS.....	1525
12.3.3 ADM_CONSTRAINTS.....	1525
12.3.4 ADM_DATA_FILES.....	1526
12.3.5 ADM_HIST_SNAPSHOT.....	1526
12.3.6 ADM_HIST_SQL_PLAN.....	1527
12.3.7 ADM_HIST_SQLSTAT.....	1527
12.3.8 ADM_IND_COLUMNS.....	1528
12.3.9 ADM_IND_EXPRESSIONS.....	1528
12.3.10 ADM_IND_PARTITIONS.....	1529
12.3.11 ADM_INDEXES.....	1530
12.3.12 ADM_OBJECTS.....	1530
12.3.13 ADM_PART_INDEXES.....	1531
12.3.14 ADM_PART_TABLES.....	1532
12.3.15 ADM_PROCEDURES.....	1532
12.3.16 ADM_SCHEDULER_JOBS.....	1533
12.3.17 ADM_SEQUENCES.....	1534
12.3.18 ADM_SOURCE.....	1535

12.3.19	ADM_SYNONYMS.....	1535
12.3.20	ADM_TAB_COLUMNS.....	1536
12.3.21	ADM_TAB_COMMENTS.....	1537
12.3.22	ADM_TAB_PARTITIONS.....	1538
12.3.23	ADM_TABLES.....	1538
12.3.24	ADM_TABLESPACES.....	1539
12.3.25	ADM_TRIGGERS.....	1539
12.3.26	ADM_TYPE_ATTRS.....	1540
12.3.27	ADM_USERS.....	1541
12.3.28	ADM_VIEWS.....	1541
12.3.29	COMM_CLIENT_INFO.....	1541
12.3.30	DB_ALL_TABLES.....	1542
12.3.31	DB_COL_COMMENTS.....	1542
12.3.32	DB_CONS_COLUMNS.....	1543
12.3.33	DB_CONSTRAINTS.....	1543
12.3.34	DB_DEPENDENCIES.....	1544
12.3.35	DB_IND_COLUMNS.....	1545
12.3.36	DB_IND_EXPRESSIONS.....	1545
12.3.37	DB_INDEXES.....	1546
12.3.38	DB_OBJECTS.....	1546
12.3.39	DB_PROCEDURES.....	1547
12.3.40	DB_SEQUENCES.....	1547
12.3.41	DB_SOURCE.....	1548
12.3.42	DB_SYNONYMS.....	1548
12.3.43	DB_TAB_COLUMNS.....	1549
12.3.44	DB_TAB_COMMENTS.....	1550
12.3.45	DB_TABLES.....	1550
12.3.46	DB_TRIGGERS.....	1551
12.3.47	DB_USERS.....	1551
12.3.48	DB_VIEWS.....	1552
12.3.49	DV_SESSION_LONGOPS.....	1552
12.3.50	DV_SESSIONS.....	1552
12.3.51	GET_GLOBAL_PREPARED_XACTS.....	1553
12.3.52	GLOBAL_BAD_BLOCK_INFO.....	1553
12.3.53	GLOBAL_CLEAR_BAD_BLOCK_INFO.....	1554
12.3.54	GLOBAL_COMM_CLIENT_INFO.....	1555
12.3.55	GLOBAL_STAT_HOTKEYS_INFO.....	1555
12.3.56	GLOBAL_WAL_SENDER_STATUS.....	1556
12.3.57	GS_ALL_CONTROL_GROUP_INFO.....	1560
12.3.58	GS_AUDITING.....	1561
12.3.59	GS_AUDITING_ACCESS.....	1561
12.3.60	GS_AUDITING_PRIVILEGE.....	1562

12.3.61 GS_CLUSTER_RESOURCE_INFO.....	1562
12.3.62 GS_DB_PRIVILEGES.....	1563
12.3.63 GS_GET_CONTROL_GROUP_INFO.....	1563
12.3.64 GS_GSC_MEMORY_DETAIL.....	1565
12.3.65 GS_LABELS.....	1565
12.3.66 GS_LSC_MEMORY_DETAIL.....	1566
12.3.67 GS_MASKING.....	1566
12.3.68 GS_MATVIEWS.....	1567
12.3.69 GS_SQL_COUNT.....	1567
12.3.70 GS_STAT_DB_CU.....	1569
12.3.71 GS_STAT_SESSION_CU.....	1569
12.3.72 GS_TOTAL_NODEGROUP_MEMORY_DETAIL.....	1570
12.3.73 GV_SESSION.....	1570
12.3.74 MPP_TABLES.....	1572
12.3.75 MY_COL_COMMENTS.....	1572
12.3.76 MY_CONS_COLUMNS.....	1573
12.3.77 MY_CONSTRAINTS.....	1573
12.3.78 MY_IND_COLUMNS.....	1574
12.3.79 MY_IND_EXPRESSIONS.....	1574
12.3.80 MY_IND_PARTITIONS.....	1575
12.3.81 MY_INDEXES.....	1576
12.3.82 MY_JOBS.....	1576
12.3.83 MY_OBJECTS.....	1577
12.3.84 MY_PART_INDEXES.....	1578
12.3.85 MY_PART_TABLES.....	1579
12.3.86 MY_PROCEDURES.....	1579
12.3.87 MY_SEQUENCES.....	1580
12.3.88 MY_SOURCE.....	1580
12.3.89 MY_SYNONYMS.....	1581
12.3.90 MY_TAB_COLUMNS.....	1582
12.3.91 MY_TAB_COMMENTS.....	1582
12.3.92 MY_TAB_PARTITIONS.....	1583
12.3.93 MY_TABLES.....	1583
12.3.94 MY_TRIGGERS.....	1584
12.3.95 MY_VIEWS.....	1585
12.3.96 PG_COMM_DELAY.....	1585
12.3.97 PG_COMM_RECV_STREAM.....	1585
12.3.98 PG_COMM_SEND_STREAM.....	1587
12.3.99 PG_COMM_STATUS.....	1588
12.3.100 PG_CONTROL_GROUP_CONFIG.....	1589
12.3.101 PG_CURSORS.....	1589
12.3.102 PG_EXT_STATS.....	1589

12.3.103 PG_GET_INVALID_BACKENDS.....	1592
12.3.104 PG_GET_SENDERS_CATCHUP_TIME.....	1592
12.3.105 PG_GROUP.....	1593
12.3.106 PG_INDEXES.....	1593
12.3.107 PG_LOCKS.....	1594
12.3.108 PG_NODE_ENV.....	1596
12.3.109 PG_OS_THREADS.....	1596
12.3.110 PG_POOLER_STATUS.....	1596
12.3.111 PG_PREPARED_STATEMENTS.....	1597
12.3.112 PG_PREPARED_XACTS.....	1598
12.3.113 PG_REPLICATION_ORIGIN_STATUS.....	1599
12.3.114 PG_REPLICATION_SLOTS.....	1599
12.3.115 PG_RLSPOICIES.....	1600
12.3.116 PG_ROLES.....	1601
12.3.117 PG_RULES.....	1603
12.3.118 PG_RUNNING_XACTS.....	1604
12.3.119 PG_SECLABELS.....	1604
12.3.120 PG_SETTINGS.....	1605
12.3.121 PG_SHADOW.....	1606
12.3.122 PG_SHARED_MEMORY_DETAIL.....	1608
12.3.123 PG_STATS.....	1609
12.3.124 PG_STAT_ACTIVITY.....	1611
12.3.125 PG_STAT_ACTIVITY_NG.....	1615
12.3.126 PG_STAT_ALL_INDEXES.....	1618
12.3.127 PG_STAT_ALL_TABLES.....	1619
12.3.128 PG_STAT_BAD_BLOCK.....	1620
12.3.129 PG_STAT_BGWRITER.....	1621
12.3.130 PG_STAT_DATABASE.....	1622
12.3.131 PG_STAT_DATABASE_CONFLICTS.....	1623
12.3.132 PG_STAT_REPLICATION.....	1624
12.3.133 PG_STAT_SYS_INDEXES.....	1626
12.3.134 PG_STAT_SYS_TABLES.....	1627
12.3.135 PG_STAT_USER_FUNCTIONS.....	1628
12.3.136 PG_STAT_USER_INDEXES.....	1629
12.3.137 PG_STAT_USER_TABLES.....	1629
12.3.138 PG_STAT_XACT_ALL_TABLES.....	1631
12.3.139 PG_STAT_XACT_SYS_TABLES.....	1631
12.3.140 PG_STAT_XACT_USER_FUNCTIONS.....	1632
12.3.141 PG_STAT_XACT_USER_TABLES.....	1633
12.3.142 PG_STATIO_ALL_INDEXES.....	1633
12.3.143 PG_STATIO_ALL_SEQUENCES.....	1634
12.3.144 PG_STATIO_ALL_TABLES.....	1634

12.3.145 PG_STATIO_SYS_INDEXES.....	1635
12.3.146 PG_STATIO_SYS_SEQUENCES.....	1635
12.3.147 PG_STATIO_SYS_TABLES.....	1636
12.3.148 PG_STATIO_USER_INDEXES.....	1637
12.3.149 PG_STATIO_USER_SEQUENCES.....	1637
12.3.150 PG_STATIO_USER_TABLES.....	1637
12.3.151 PG_TABLES.....	1638
12.3.152 PG_TDE_INFO.....	1639
12.3.153 PG_TIMEZONE_ABBREVS.....	1640
12.3.154 PG_TIMEZONE_NAMES.....	1640
12.3.155 PG_TOTAL_MEMORY_DETAIL.....	1640
12.3.156 PG_TOTAL_USER_RESOURCE_INFO.....	1641
12.3.157 PG_TOTAL_USER_RESOURCE_INFO_OID.....	1642
12.3.158 PG_USER.....	1643
12.3.159 PG_USER_MAPPINGS.....	1645
12.3.160 PG_VARIABLE_INFO.....	1646
12.3.161 PG_VIEWS.....	1646
12.3.162 PGXC_COMM_DELAY.....	1647
12.3.163 PGXC_COMM_RECV_STREAM.....	1647
12.3.164 PGXC_COMM_SEND_STREAM.....	1649
12.3.165 PGXC_COMM_STATUS.....	1650
12.3.166 PGXC_GET_STAT_ALL_TABLES.....	1651
12.3.167 PGXC_GET_TABLE_SKEWNESS.....	1651
12.3.168 PGXC_NODE_ENV.....	1652
12.3.169 PGXC_OS_THREADS.....	1652
12.3.170 PGXC_PREPARED_XACTS.....	1653
12.3.171 PGXC_RUNNING_XACTS.....	1653
12.3.172 PGXC_SQL_COUNT.....	1654
12.3.173 PGXC_STAT_ACTIVITY.....	1655
12.3.174 PGXC_STAT_BAD_BLOCK.....	1658
12.3.175 PGXC_THREAD_WAIT_STATUS.....	1658
12.3.176 PGXC_TOTAL_MEMORY_DETAIL.....	1660
12.3.177 PGXC_VARIABLE_INFO.....	1661
12.3.178 PLAN_TABLE.....	1662
12.3.179 PV_FILE_STAT.....	1663
12.3.180 PV_INSTANCE_TIME.....	1664
12.3.181 PV_OS_RUN_INFO.....	1665
12.3.182 PV_REDO_STAT.....	1665
12.3.183 PV_SESSION_MEMORY.....	1666
12.3.184 PV_SESSION_MEMORY_CONTEXT.....	1666
12.3.185 PV_SESSION_MEMORY_DETAIL.....	1667
12.3.186 PV_SESSION_STAT.....	1668

12.3.187 PV_SESSION_TIME.....	1668
12.3.188 PV_THREAD_MEMORY_CONTEXT.....	1669
12.3.189 PV_TOTAL_MEMORY_DETAIL.....	1670
12.3.190 SYS_DUMMY.....	1671
13 Schemas.....	1673
13.1 Information Schema.....	1675
13.1.1 _PG_FOREIGN_DATA_WRAPPERS.....	1676
13.1.2 _PG_FOREIGN_SERVERS.....	1676
13.1.3 _PG_FOREIGN_TABLE_COLUMNS.....	1677
13.1.4 _PG_FOREIGN_TABLES.....	1677
13.1.5 _PG_USER_MAPPINGS.....	1678
13.1.6 INFORMATION_SCHEMA_CATALOG_NAME.....	1679
13.2 DBE_PERF Schema.....	1679
13.2.1 OS.....	1679
13.2.1.1 OS_RUNTIME.....	1679
13.2.1.2 GLOBAL_OS_RUNTIME.....	1680
13.2.1.3 OS_THREADS.....	1680
13.2.1.4 GLOBAL_OS_THREADS.....	1681
13.2.2 Instance.....	1681
13.2.2.1 INSTANCE_TIME.....	1681
13.2.2.2 GLOBAL_INSTANCE_TIME.....	1682
13.2.3 Memory.....	1682
13.2.3.1 MEMORY_NODE_DETAIL.....	1682
13.2.3.2 GLOBAL_MEMORY_NODE_DETAIL.....	1683
13.2.3.3 MEMORY_NODE_NG_DETAIL.....	1685
13.2.3.4 SHARED_MEMORY_DETAIL.....	1685
13.2.3.5 GLOBAL_SHARED_MEMORY_DETAIL.....	1686
13.2.3.6 TRACK_MEMORY_CONTEXT_DETAIL.....	1686
13.2.4 File.....	1687
13.2.4.1 FILE_IOSTAT.....	1687
13.2.4.2 SUMMARY_FILE_IOSTAT.....	1688
13.2.4.3 GLOBAL_FILE_IOSTAT.....	1688
13.2.4.4 FILE_REDO_IOSTAT.....	1689
13.2.4.5 SUMMARY_FILE_REDO_IOSTAT.....	1690
13.2.4.6 GLOBAL_FILE_REDO_IOSTAT.....	1690
13.2.4.7 LOCAL_REL_IOSTAT.....	1691
13.2.4.8 GLOBAL_REL_IOSTAT.....	1691
13.2.4.9 SUMMARY_REL_IOSTAT.....	1692
13.2.5 Object.....	1692
13.2.5.1 STAT_USER_TABLES.....	1692
13.2.5.2 SUMMARY_STAT_USER_TABLES.....	1694
13.2.5.3 GLOBAL_STAT_USER_TABLES.....	1695

13.2.5.4	STAT_USER_INDEXES.....	1696
13.2.5.5	SUMMARY_STAT_USER_INDEXES.....	1697
13.2.5.6	GLOBAL_STAT_USER_INDEXES.....	1697
13.2.5.7	STAT_SYS_TABLES.....	1698
13.2.5.8	SUMMARY_STAT_SYS_TABLES.....	1699
13.2.5.9	GLOBAL_STAT_SYS_TABLES.....	1701
13.2.5.10	STAT_SYS_INDEXES.....	1702
13.2.5.11	SUMMARY_STAT_SYS_INDEXES.....	1702
13.2.5.12	GLOBAL_STAT_SYS_INDEXES.....	1703
13.2.5.13	STAT_ALL_TABLES.....	1703
13.2.5.14	SUMMARY_STAT_ALL_TABLES.....	1705
13.2.5.15	GLOBAL_STAT_ALL_TABLES.....	1706
13.2.5.16	STAT_ALL_INDEXES.....	1707
13.2.5.17	SUMMARY_STAT_ALL_INDEXES.....	1708
13.2.5.18	GLOBAL_STAT_ALL_INDEXES.....	1708
13.2.5.19	STAT_DATABASE.....	1709
13.2.5.20	SUMMARY_STAT_DATABASE.....	1710
13.2.5.21	GLOBAL_STAT_DATABASE.....	1712
13.2.5.22	STAT_DATABASE_CONFLICTS.....	1713
13.2.5.23	SUMMARY_STAT_DATABASE_CONFLICTS.....	1714
13.2.5.24	GLOBAL_STAT_DATABASE_CONFLICTS.....	1714
13.2.5.25	STAT_XACT_ALL_TABLES.....	1715
13.2.5.26	SUMMARY_STAT_XACT_ALL_TABLES.....	1715
13.2.5.27	GLOBAL_STAT_XACT_ALL_TABLES.....	1716
13.2.5.28	STAT_XACT_SYS_TABLES.....	1717
13.2.5.29	SUMMARY_STAT_XACT_SYS_TABLES.....	1717
13.2.5.30	GLOBAL_STAT_XACT_SYS_TABLES.....	1718
13.2.5.31	STAT_XACT_USER_TABLES.....	1719
13.2.5.32	SUMMARY_STAT_XACT_USER_TABLES.....	1719
13.2.5.33	GLOBAL_STAT_XACT_USER_TABLES.....	1720
13.2.5.34	STAT_XACT_USER_FUNCTIONS.....	1721
13.2.5.35	SUMMARY_STAT_XACT_USER_FUNCTIONS.....	1721
13.2.5.36	GLOBAL_STAT_XACT_USER_FUNCTIONS.....	1721
13.2.5.37	STAT_BAD_BLOCK.....	1722
13.2.5.38	SUMMARY_STAT_BAD_BLOCK.....	1722
13.2.5.39	GLOBAL_STAT_BAD_BLOCK.....	1723
13.2.5.40	STAT_USER_FUNCTIONS.....	1723
13.2.5.41	SUMMARY_STAT_USER_FUNCTIONS.....	1724
13.2.5.42	GLOBAL_STAT_USER_FUNCTIONS.....	1724
13.2.6	Workload.....	1725
13.2.6.1	WORKLOAD_SQL_COUNT.....	1725
13.2.6.2	SUMMARY_WORKLOAD_SQL_COUNT.....	1726

13.2.6.3 WORKLOAD_TRANSACTION.....	1726
13.2.6.4 SUMMARY_WORKLOAD_TRANSACTION.....	1727
13.2.6.5 GLOBAL_WORKLOAD_TRANSACTION.....	1728
13.2.6.6 WORKLOAD_SQL_ELAPSE_TIME.....	1729
13.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME.....	1730
13.2.6.8 USER_TRANSACTION.....	1731
13.2.6.9 GLOBAL_USER_TRANSACTION.....	1732
13.2.7 Session and Thread.....	1733
13.2.7.1 SESSION_STAT.....	1733
13.2.7.2 GLOBAL_SESSION_STAT.....	1733
13.2.7.3 SESSION_TIME.....	1734
13.2.7.4 GLOBAL_SESSION_TIME.....	1734
13.2.7.5 SESSION_MEMORY.....	1734
13.2.7.6 GLOBAL_SESSION_MEMORY.....	1735
13.2.7.7 SESSION_MEMORY_DETAIL.....	1735
13.2.7.8 GLOBAL_SESSION_MEMORY_DETAIL.....	1736
13.2.7.9 THREAD_WAIT_STATUS.....	1736
13.2.7.10 GLOBAL_THREAD_WAIT_STATUS.....	1737
13.2.7.11 LOCAL_THREADPOOL_STATUS.....	1738
13.2.7.12 GLOBAL_THREADPOOL_STATUS.....	1740
13.2.7.13 SESSION_CPU_RUNTIME.....	1740
13.2.7.14 SESSION_MEMORY_RUNTIME.....	1740
13.2.7.15 LOCAL_ACTIVE_SESSION.....	1741
13.2.7.16 GLOBAL_ACTIVE_SESSION.....	1743
13.2.8 Transaction.....	1745
13.2.8.1 TRANSACTIONS_RUNNING_XACTS.....	1745
13.2.8.2 SUMMARY_TRANSACTIONS_RUNNING_XACTS.....	1746
13.2.8.3 GLOBAL_TRANSACTIONS_RUNNING_XACTS.....	1746
13.2.8.4 TRANSACTIONS_PREPARED_XACTS.....	1747
13.2.8.5 SUMMARY_TRANSACTIONS_PREPARED_XACTS.....	1747
13.2.8.6 GLOBAL_TRANSACTIONS_PREPARED_XACTS.....	1748
13.2.9 Query.....	1748
13.2.9.1 STATEMENT.....	1748
13.2.9.2 SUMMARY_STATEMENT.....	1752
13.2.9.3 STATEMENT_COUNT.....	1755
13.2.9.4 GLOBAL_STATEMENT_COUNT.....	1756
13.2.9.5 SUMMARY_STATEMENT_COUNT.....	1758
13.2.9.6 STATEMENT_RESPONSETIME_PERCENTILE.....	1759
13.2.9.7 STATEMENT_HISTORY.....	1760
13.2.10 Cache and I/O.....	1764
13.2.10.1 STATIO_USER_TABLES.....	1764
13.2.10.2 SUMMARY_STATIO_USER_TABLES.....	1765

13.2.10.3 GLOBAL_STATIO_USER_TABLES.....	1766
13.2.10.4 STATIO_USER_INDEXES.....	1766
13.2.10.5 SUMMARY_STATIO_USER_INDEXES.....	1767
13.2.10.6 GLOBAL_STATIO_USER_INDEXES.....	1767
13.2.10.7 STATIO_USER_SEQUENCES.....	1768
13.2.10.8 SUMMARY_STATIO_USER_SEQUENCES.....	1768
13.2.10.9 GLOBAL_STATIO_USER_SEQUENCES.....	1769
13.2.10.10 STATIO_SYS_TABLES.....	1769
13.2.10.11 SUMMARY_STATIO_SYS_TABLES.....	1770
13.2.10.12 GLOBAL_STATIO_SYS_TABLES.....	1770
13.2.10.13 STATIO_SYS_INDEXES.....	1771
13.2.10.14 SUMMARY_STATIO_SYS_INDEXES.....	1772
13.2.10.15 GLOBAL_STATIO_SYS_INDEXES.....	1772
13.2.10.16 STATIO_SYS_SEQUENCES.....	1773
13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES.....	1773
13.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES.....	1773
13.2.10.19 STATIO_ALL_TABLES.....	1774
13.2.10.20 SUMMARY_STATIO_ALL_TABLES.....	1775
13.2.10.21 GLOBAL_STATIO_ALL_TABLES.....	1775
13.2.10.22 STATIO_ALL_INDEXES.....	1776
13.2.10.23 SUMMARY_STATIO_ALL_INDEXES.....	1776
13.2.10.24 GLOBAL_STATIO_ALL_INDEXES.....	1777
13.2.10.25 STATIO_ALL_SEQUENCES.....	1777
13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES.....	1778
13.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES.....	1778
13.2.10.28 GLOBAL_STAT_DB_CU.....	1779
13.2.10.29 GLOBAL_STAT_SESSION_CU.....	1779
13.2.11 Communications Library.....	1779
13.2.11.1 COMM_DELAY.....	1779
13.2.11.2 GLOBAL_COMM_DELAY.....	1780
13.2.11.3 COMM_RECV_STREAM.....	1781
13.2.11.4 GLOBAL_COMM_RECV_STREAM.....	1782
13.2.11.5 COMM_SEND_STREAM.....	1782
13.2.11.6 GLOBAL_COMM_SEND_STREAM.....	1783
13.2.11.7 COMM_STATUS.....	1784
13.2.11.8 GLOBAL_COMM_STATUS.....	1785
13.2.12 Utility.....	1786
13.2.12.1 REPLICATION_STAT.....	1786
13.2.12.2 GLOBAL_REPLICATION_STAT.....	1787
13.2.12.3 REPLICATION_SLOTS.....	1788
13.2.12.4 GLOBAL_REPLICATION_SLOTS.....	1789
13.2.12.5 BGWRITER_STAT.....	1790

13.2.12.6 GLOBAL_BGWRITER_STAT.....	1791
13.2.12.7 POOLER_STATUS.....	1792
13.2.12.8 GLOBAL_COMM_CHECK_CONNECTION_STATUS.....	1793
13.2.12.9 GLOBAL_CKPT_STATUS.....	1793
13.2.12.10 GLOBAL_DOUBLE_WRITE_STATUS.....	1794
13.2.12.11 GLOBAL_PAGEWRITER_STATUS.....	1795
13.2.12.12 GLOBAL_POOLER_STATUS.....	1795
13.2.12.13 GLOBAL_RECORD_RESET_TIME.....	1796
13.2.12.14 GLOBAL_REDO_STATUS.....	1796
13.2.12.15 GLOBAL_RECOVERY_STATUS.....	1798
13.2.12.16 CLASS_VITAL_INFO.....	1799
13.2.12.17 USER_LOGIN.....	1799
13.2.12.18 SUMMARY_USER_LOGIN.....	1800
13.2.12.19 GLOBAL_GET_BGWRITER_STATUS.....	1800
13.2.12.20 GLOBAL_SINGLE_FLUSH_DW_STATUS.....	1800
13.2.12.21 GLOBAL_CANDIDATE_STATUS.....	1801
13.2.13 Lock.....	1802
13.2.13.1 LOCKS.....	1802
13.2.13.2 GLOBAL_LOCKS.....	1803
13.2.14 Wait Event.....	1805
13.2.14.1 WAIT_EVENTS.....	1805
13.2.14.2 GLOBAL_WAIT_EVENTS.....	1806
13.2.15 Configuration.....	1806
13.2.15.1 CONFIG_SETTINGS.....	1806
13.2.15.2 GLOBAL_CONFIG_SETTINGS.....	1807
13.2.16 Operator.....	1808
13.2.16.1 OPERATOR_HISTORY_TABLE.....	1809
13.2.16.2 OPERATOR_HISTORY.....	1810
13.2.16.3 OPERATOR_RUNTIME.....	1810
13.2.16.4 GLOBAL_OPERATOR_HISTORY.....	1812
13.2.16.5 GLOBAL_OPERATOR_HISTORY_TABLE.....	1813
13.2.16.6 GLOBAL_OPERATOR_RUNTIME.....	1814
13.2.17 Global Plan Cache.....	1815
13.2.17.1 LOCAL_PLANCACHE_STATUS.....	1815
13.2.17.2 GLOBAL_PLANCACHE_STATUS.....	1816
13.2.18 RTO & RPO.....	1816
13.2.18.1 global_rto_status.....	1816
13.2.18.2 global_streaming_hadr_rto_and_rpo_stat.....	1816
13.2.19 Deprecated Views.....	1817
13.2.19.1 Query.....	1817
13.2.19.2 Global Plancache.....	1817
14 Logical Replication.....	1818

14.1 Logical Decoding.....	1818
14.1.1 Overview.....	1818
14.1.2 Logical Decoding Options.....	1822
14.1.3 Logical Decoding by SQL Function Interfaces.....	1827
14.1.4 Replicating Data Using the Logical Replication Tool.....	1828
15 GTM Mode.....	1829
16 Materialized View.....	1832
16.1 Complete-refresh Materialized View.....	1832
16.1.1 Overview.....	1832
16.1.2 Usage.....	1832
16.1.3 Support and Constraints.....	1833
16.2 Fast-refresh Materialized View.....	1833
16.2.1 Overview.....	1834
16.2.2 Usage.....	1834
16.2.3 Support and Constraints.....	1835
17 Error Log Reference.....	1837
17.1 Kernel Error Information.....	1837
17.2 CM Error Information.....	1861
18 Configuring GUC Parameters.....	1870
18.1 Viewing Parameter Values.....	1870
18.2 Resetting Parameters.....	1871
18.3 GUC Parameters.....	1873
18.3.1 GUC Parameter Usage.....	1873
18.3.2 File Location.....	1874
18.3.3 Connection and Authentication.....	1875
18.3.3.1 Connection Settings.....	1875
18.3.3.2 Security and Authentication (postgresql.conf).....	1884
18.3.3.3 Communication Library Parameters.....	1894
18.3.4 Resource Consumption.....	1904
18.3.4.1 Memory.....	1904
18.3.4.2 Disk Space.....	1918
18.3.4.3 Kernel Resource Usage.....	1919
18.3.4.4 Cost-based Vacuum Delay.....	1920
18.3.4.5 Background Writer.....	1922
18.3.4.6 Asynchronous I/O.....	1925
18.3.5 Parallel Data Import.....	1926
18.3.6 Write Ahead Log.....	1928
18.3.6.1 Settings.....	1928
18.3.6.2 Checkpoints.....	1936
18.3.6.3 Log Replay.....	1938
18.3.6.4 Archiving.....	1941

18.3.7 HA Replication.....	1942
18.3.7.1 Sending Server.....	1942
18.3.7.2 Primary Server.....	1949
18.3.7.3 Standby Server.....	1954
18.3.8 Query Planning.....	1958
18.3.8.1 Optimizer Method Configuration.....	1958
18.3.8.2 Optimizer Cost Constants.....	1968
18.3.8.3 Genetic Query Optimizer.....	1971
18.3.8.4 Other Optimizer Options.....	1973
18.3.9 Error Reporting and Logging.....	1992
18.3.9.1 Logging Destination.....	1992
18.3.9.2 Logging Time.....	1996
18.3.9.3 Logging Content.....	1999
18.3.9.4 Using CSV Log Output.....	2008
18.3.10 Alarm Detection.....	2010
18.3.11 Statistics During the Database Running.....	2011
18.3.11.1 Query and Index Statistics Collector.....	2011
18.3.11.2 Performance Statistics.....	2015
18.3.11.3 Hotspot Key Statistics.....	2016
18.3.12 Automatic Vacuuming.....	2016
18.3.13 Default Settings of Client Connection.....	2021
18.3.13.1 Statement Behavior.....	2021
18.3.13.2 Locale and Formatting.....	2027
18.3.13.3 Other Default Parameters.....	2032
18.3.14 Lock Management.....	2033
18.3.15 Version and Platform Compatibility.....	2038
18.3.15.1 Compatibility with Earlier Versions.....	2038
18.3.15.2 Platform and Client Compatibility.....	2041
18.3.16 Fault Tolerance.....	2055
18.3.17 Connection Pool Parameters.....	2057
18.3.18 Cluster Transaction Parameters.....	2062
18.3.19 Dual-Cluster Replication Parameters.....	2071
18.3.20 Developer Options.....	2074
18.3.21 Auditing.....	2081
18.3.21.1 Audit Switch.....	2081
18.3.21.2 User and Permission Audit.....	2085
18.3.21.3 Operation Auditing.....	2086
18.3.22 Transaction Monitoring.....	2093
18.3.23 CM Parameters.....	2094
18.3.23.1 CM Agent Parameters.....	2094
18.3.23.2 CM Server Parameters.....	2101
18.3.24 GTM Parameters.....	2111

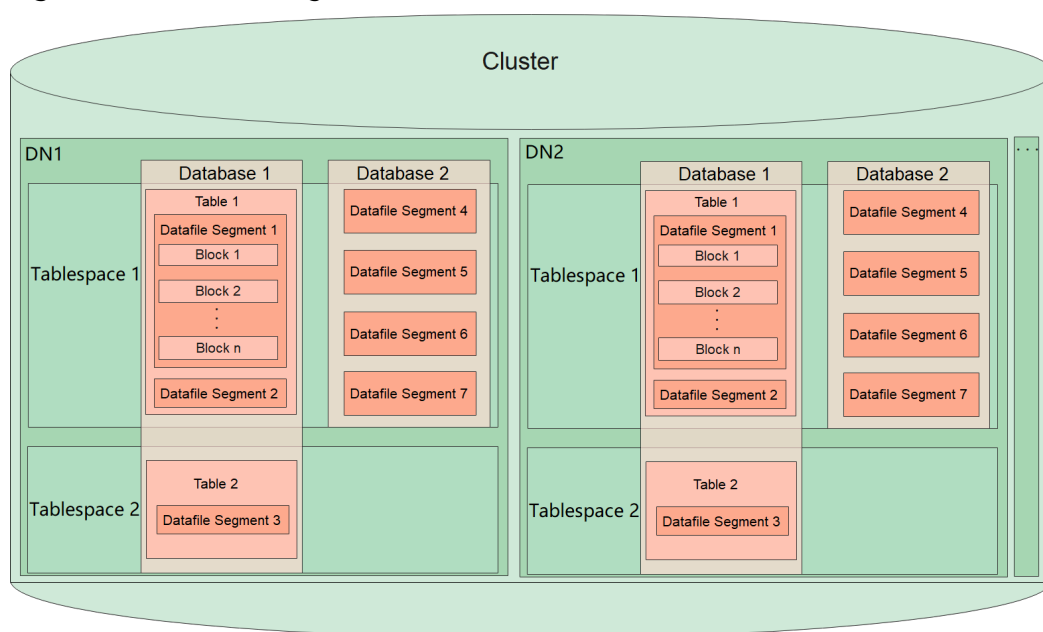
18.3.25 Upgrade Parameters.....	2119
18.3.26 Miscellaneous Parameters.....	2120
18.3.27 Wait Event.....	2124
18.3.28 Query.....	2124
18.3.29 System Performance Snapshot.....	2130
18.3.30 Security Configuration.....	2132
18.3.31 HyperLogLog.....	2133
18.3.32 User-defined Functions.....	2135
18.3.33 Scheduled Task.....	2136
18.3.34 Thread Pool.....	2137
18.3.35 Backup and Restoration.....	2141
18.3.36 AI Features.....	2142
18.3.37 Global SysCache Parameters.....	2142
18.3.38 Reserved Parameters.....	2143

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

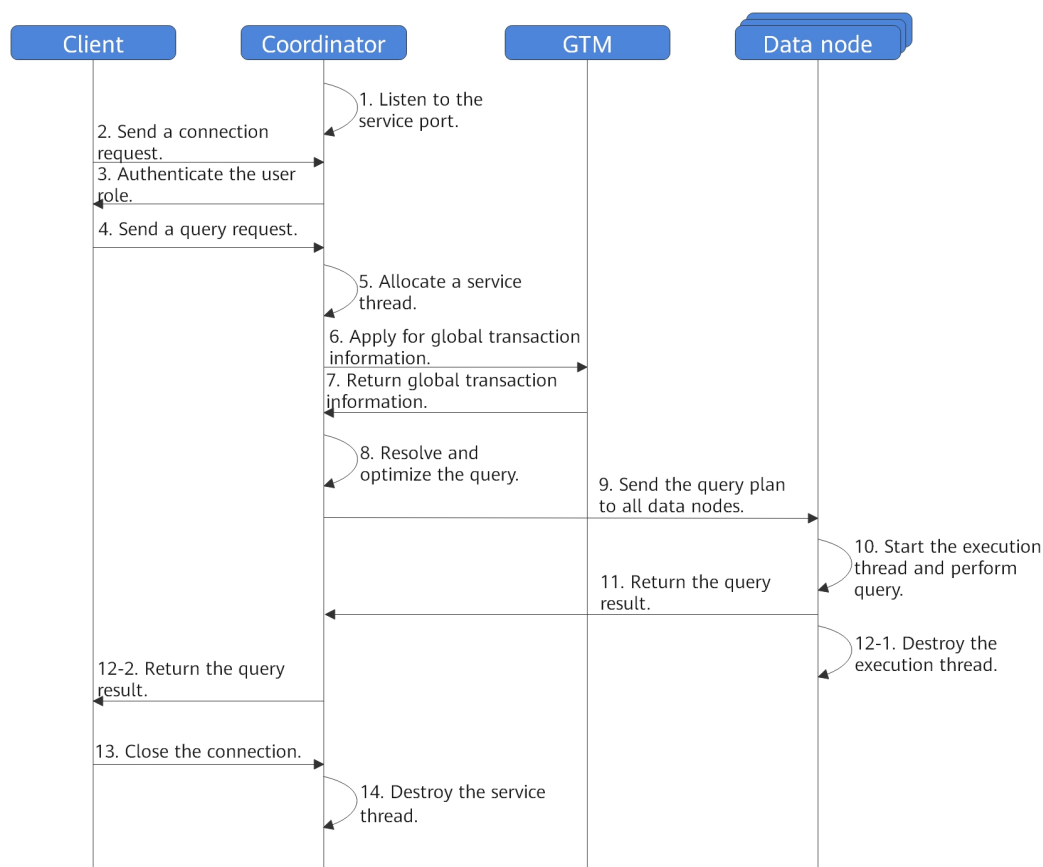
- A tablespace is a directory that stores physical files of databases. Multiple tablespaces may exist in a cluster. Each tablespace can contain files belonging to different databases.
- Each database manages various data objects and is isolated from each other. Objects managed by a database can be distributed 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 data file segments storing the data of the same table must be in the same tablespace.
- Block: Basic unit of database management. Its default size is 8 KB.
- Data can be distributed on DNs in replication, hash, range, or list mode. You can specify a mode when creating a table.

1.2 Query Request Handling Process

The GaussDB service response process involves components such as the client, Coordinator, GTM, and DNs.

Figure 1-2 shows the specific 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. A GaussDB database supports the following transaction isolation levels: READ COMMITTED, READ UNCOMMITTED (not recommended), REPEATABLE READ, and SERIALIZABLE. SERIALIZABLE is equivalent to REPEATABLE READ.

Controlling Transactions

The following describes transaction operations supported by the database:

- 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 use the COMMIT or END syntax to commit all operations of a transaction. 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 data is modified using SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY in the transaction.

- **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. It can also access the data modifications in its transaction, regardless of whether they have been committed. Note that different database snapshots may be returned 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. Pay attention to whether a single statement always accesses completely 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.

- **READ UNCOMMITTED:** This level is not recommended, because it may result in data inconsistency. This level can be used for reading data in an emergency, such as when a coordinator node (CN) fails to be recovered from a fault, to bypass the congestion caused by data inconsistency between the GTM and CN/DN. Do not use this level for write transactions, as this may lead to data inconsistency.
- **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**.

1.4 Concepts

Database

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.

Tablespace

In GaussDB, a tablespace is a directory that stores physical files of databases. Multiple tablespaces may exist in a cluster. Files are physically isolated using tablespaces and managed by a file system.

Schema

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.

User and Role

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. In GaussDB, the difference between roles and users is that a role does not have the

LOGIN permission by default. In GaussDB, one user can have only one role, but you can put a user's role under a parent role to grant multiple permissions to the user.

Transaction

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, MVCC stores various versions of historical data together with the current tuple version. GaussDB uses the VACUUM thread instead of rollback segments to routinely delete various versions of historical data. Generally, you do not need to pay special attention to the VACUUM thread unless you need to optimize the performance. In addition, GaussDB automatically commits transactions for single-statement queries (without using statements such as BEGIN to explicitly start a transaction block).

2 Database Security Management

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, [the 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 administrator 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, run [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, for example, making tables read-only to themselves or others.

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](#).

You are advised not to modify the permissions on system catalogs or system views.

2.1.2 Administrator

Initial User

The account automatically generated during the cluster installation is called the initial user. The initial user has the highest-level permissions in the system, including SYSADMIN, CREATEROLE, AUDITADMIN, MONADMIN, OPRADMIN, and POLADMIN, 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 any SQL operations, such as upgrade, scaling, 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 **dbe_perf** schema or the permission to use Roach to perform backup and restoration.

To create a database administrator, connect to the database as an administrator and run the **CREATE USER** or **ALTER USER** statement with **SYSADMIN** specified.

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

or

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

To run the **ALTER USER** statement, the user must exist.

Security Administrator

A security administrator is an account with the CREATEROLE attribute. It has the permission to create, modify, and delete users or roles, and grant or revoke the permission of any non-system administrator, built-in role, permanent user, or O&M administrator.

If you want to create a security administrator when separation of duties is disabled, connect to the database as a system administrator or security administrator. When 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.

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

or

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

To run the **ALTER USER** statement, 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 when separation of duties is disabled, connect to the database as a system administrator or security administrator. When 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.

```
openGauss=# CREATE USER auditadmin WITH AUDITADMIN password '*****';
```

or

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

To run the **ALTER USER** statement, 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 **dbf_perf** schema. The monitor administrator can also grant or revoke object permissions in the **dbf_perf** schema.

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

```
openGauss=# CREATE USER monadmin WITH MONADMIN password '*****';
```

or

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

To run the **ALTER USER** statement, 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 run the **CREATE USER** or **ALTER USER** statement with **OPRADMIN** specified.

```
openGauss=# CREATE USER opradmin WITH OPRADMIN password '*****';
```

or

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

To run the **ALTER USER** statement, 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 run the **CREATE USER** or **ALTER USER** statement with **POLADMIN** specified.

```
openGauss=# CREATE USER poladmin WITH POLADMIN password '*****';
```

or

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

To run the **ALTER USER** statement, the user must exist.

2.1.3 Separation of Duties

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

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 is enabled, 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.

To enable separation of duties, set GUC parameter **enableSeparationOfDuty** to **on**.

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 dbperf .	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	N/A Has all permissions.	N/A	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, a system administrator can still access the two system catalogs to obtain sensitive information in 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 non-**separation-of-duties** scenarios, 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 scenarios, 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 on 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 a user **joe** and set the **CREATEDB** attribute for the user.

```
openGauss=# CREATE USER joe WITH CREATEDB PASSWORD '*****';  
CREATE ROLE
```
- To create a system administrator, use the **CREATE USER** statement with the **SYSADMIN** parameter.
- 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.
openGauss=# SELECT * FROM pg_user;
- To view user attributes, query the system catalog **PG_AUTHID**.
openGauss=# SELECT * FROM pg_authid;

Permanent User

GaussDB provides the permanent user solution. That is, create a permanent user with the **PERSISTENCE** attribute.

```
openGauss=# 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 role members.

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 the 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 delete a role, use **DROP ROLE**. DROP ROLE deletes only a role, rather than role members.

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 Permission description of built-in roles

Roles	Permission
gs_role_copy_files	Permission to run the copy... to/from filename command. However, the GUC parameter enable_copy_server_files must be set first to enable the function of copying server files.
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. 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 PERSISTENCE user.
gs_role_pldebugger	Permission to debug functions in db_pldebugger .
gs_role_directory_create	Permission to create directory objects. However, this role needs to enable the GUC parameter enable_access_server_directory first.
gs_role_directory_drop	Permission to delete directory objects. However, this role needs to enable the GUC parameter enable_access_server_directory first.

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 or roles starting with "gs_role_" or rename existing users or roles starting with "gs_role_".
- Do not perform ALTER or DROP operation on built-in roles.
- By default, built-in roles do not have the LOGIN permission and do not have preset passwords.
- The **gsql** 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 the system administrator and 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 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 under the **pg_catalog** schema.

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 administrator 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 the schema.
- To delete a schema and its objects, use [DROP SCHEMA](#). Schema owners can delete schemas.
- 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 following 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.


```
openGauss=# 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**.
- 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.

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

Search Path

A search path is defined in the **search_path** parameter. 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**.

```
openGauss=# 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 to only to the **public** schema.

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

Run the following command to set the search path to **myschema,public** (**myschema** will be searched before **public**):

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

2.1.7 Setting 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.

```
openGauss=# GRANT USAGE ON SCHEMA tpcds TO joe;  
openGauss=# 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.

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

- b. Grant object permissions to the role by using **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**.

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

- c. Grant the role permissions to a user.

```
openGauss=# 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 Policy

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 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.  
openGauss=# CREATE USER alice PASSWORD '*****';  
openGauss=# CREATE USER bob PASSWORD '*****';  
openGauss=# CREATE USER peter PASSWORD '*****';  
  
-- Create the all_data table that contains user information.  
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));  
  
-- Insert data into the data table.  
openGauss=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');  
openGauss=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');  
openGauss=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');  
  
-- Grant the read permission for the all_data table to users alice, bob, and peter.  
openGauss=# GRANT SELECT ON all_data TO alice, bob, peter;  
  
-- Enable row-level security.  
openGauss=# 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.  
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =  
CURRENT_USER);
```

```
-- View table details.
openGauss=# \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_rls"
USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, enable_rowsecurity=true

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

openGauss=# 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.
openGauss=# SELECT * FROM public.all_data;
id | role | data
---+---+---
3 | peter | peter data
(1 row)

openGauss=# 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 the information in the statistical objects.

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](#), it 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 database connection tool running in the CLI provided by GaussDB. gsql provides basic and advanced functions of databases to facilitate user operations. This section describes how to use gsql to connect to a database. For details about how to use gsql, see "Client Tools > gsql" in *Tool Reference*.

Precautions

By default, when a client has been idle for the period specified by GUC parameter **session_timeout** after connecting to a database, the client automatically disconnects from the database. To disable the timeout setting, set GUC parameter **session_timeout** to 0.

Prerequisites

You have contacted the administrator for connection information.

Remotely Connecting to a Database

- Step 1** Contact an administrator to configure the remote connection.
- Step 2** On the host, upload the client tool package and configure environment variables for the gsql client.

1. Log in to the client.
2. Run the following command to create the **/tmp/tools** directory:
`mkdir /tmp/tools`
3. Obtain **GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-gsqr.tar.gz** from the software installation package and upload it to the **/tmp/tools** directory.

NOTE

- The software package is located where you put it before installation. Set it based on the actual situation.
 - The tool package name may vary in different OSs. Select the tool package suitable for your OS.
4. Run the following commands to decompress the package:
`cd /tmp/tools`
`tar -zxvf GaussDB-Kernel_VxxxRxxxCxx-XXXXX-64bit-gsqr.tar.gz`
 5. Set environment variables.
Run the following command to open the **~/.bashrc** file:
`vi ~/.bashrc`
Enter the following content and run **:wq!** to save and exit.
`export PATH=/tmp/tools/bin:$PATH`
`export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH`
 6. Run the following command to make the environment variables take effect:
`source ~/.bashrc`

Step 3 Connect to a database.

After the database is installed, a database named **postgres** is generated by default.

```
gsqr -d postgres -h 10.10.0.11 -U jack -p 8000
Password for user jack:
```

postgres is the name of the database to be connected, **10.10.0.11** is the IP address of the server where the CN resides, **jack** is the username for logging in to the database, **8000** is the port number of the CN. The IP address and port number of the connected server can be replaced with those of the server where the DN resides.

NOTE

- In the preceding example, user **jack** is a common user created by the database initial user.
- By default, the initial user of the database is not allowed to remotely connect to the database. If Kerberos authentication is enabled in a cluster, the initial user is allowed to remotely connect to the database in the cluster.

----End

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	- Linux: Driver: GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz unixODBC source code package: http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download - Windows: Driver: GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc.tar.gz
JDBC	- Driver: GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Jdbc.tar.gz - Driver: 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.

```
openGauss=# 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 *********. Run the following command to set user **joe** as the system administrator:

```
openGauss=# 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 the database system administrator 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](#).

Background

- 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 any other tools to connect to or to perform operations on the 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 database does not allow user connections. Only the initial user and the system administrator of the database 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 by specifying its name after truncation, specify its original name to delete it, or manually delete it from the system catalogs on each node.

Procedure

Step 1 Run the following command to create a database named **db_tpcds**:

```
openGauss=# 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. To specify another tablespace, run the following statement:

```
openGauss=# 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.

```
openGauss=# \q  
gsqsl -d db_tpcds -p 8000 -U joe  
Password for user joe:  
gsqsl((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.

```
openGauss=# \l
```

- Run the following command to query the database list in the **pg_database** system catalog:

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

Step 3 Modify the database.

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

- Run the following command to rename the database:

```
openGauss=# 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 Delete a 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.

Run the following command to delete the database:

```
openGauss=# DROP DATABASE human_tpcds;  
DROP DATABASE
```

----End

3.2.3 Creating and Managing Tablespaces

Background

The administrator can use tablespaces to control the layout of disks where a database is installed. This has the following advantages:

- 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 system space is reconfigured.
- Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance.
 - A frequently used index can be placed in a disk having stable performance and high computing speed, such as a solid 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.
- The administrator can use tablespaces to set the maximum available disk space. In this way, when a partition is shared with other data, tablespaces will not occupy excessive space in the partition.
- You can use tablespaces to control the disk space occupied by data in a database. If the usage of a disk where a tablespace resides reaches 90%, the database switches to the read-only mode. It switches back to read/write mode when the disk usage becomes less than 90%.

The automatic disk check of the cluster manager (CM) is enabled by default. To enable it, run the following command:

```
gs_guc set -Z cmserver -N all -I all -c "enable_transaction_read_only = on "
```

Restart the database to make the parameter settings take effect.

- Each tablespace corresponds to a file system directory. 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.

```
openGauss=# 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 tables. The corresponding storage directory is the base directory in the global data directory.

Precautions:

- You are not advised to use user-defined tablespaces.
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.

Procedure

- Create a tablespace.
 - Run the following command to create user **jack** and set the password to *********.

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

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

```
CREATE ROLE
```
 - Run the following command to create a tablespace:

```
openGauss=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'my_tablespace/tablespace1';
```

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

```
CREATE TABLESPACE
```

fastspace is the new tablespace, and *CN/DN data directory/pg_location/my_tablespace/tablespace1* is an empty directory on which users have read and write permissions.
 - A database system administrator can run the following command to grant the permission of accessing the **fastspace** tablespace to user **jack**:

```
openGauss=# 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: Run the following command to create a table in a specified tablespace:

```
openGauss=# 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: Use **set default_tablespace** to set the default tablespace and then create a table:

```
openGauss=# SET default_tablespace = 'fastspace';
SET
openGauss=# 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.

- Use one of the following methods to query a tablespace:
 - Method 1: Check the **pg_tablespace** system catalog. Run the following command to view all the tablespaces defined by the system and users:
openGauss=# **SELECT spcname FROM pg_tablespace;**
 - Method 2: Run the following meta-command of the gsql program to query the tablespaces:
openGauss=# **\db**

- Query the tablespace usage.

- a. Query the current usage of the tablespace.

```
openGauss=# 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.

Run the following command to rename tablespace **fastspace** to **fspace**:

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

- Delete a tablespace and related data.

- Run the following command to delete user **jack**:

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

- Run the following commands to delete tables **foo** and **foo2**:

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

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

```
DROP TABLE
```

- Run the following command to delete tablespace **fspace**:

```
openGauss=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

NOTE

Only the tablespace owner or system administrator 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 saved 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

Run the following command to create a table:

```
openGauss=# 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, such as the **mytable** table, 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 a System Catalog](#).

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

The length of a character on the server and client may vary by the character sets they use. A string entered on the client will be processed based on the server's character set, so the result may differ from expected.

Table 3-2 Comparison of character set output between the client and server

Operation Procedure	Server and Client Use Same Encoding	Server and Client Use Different Encoding
No operations are performed to the string while it is saved and read.	Your expected result is returned.	If the encoding for input and output on the client is the same, your expected result is returned.
Operations (such as executing string functions) are performed to the string while it is saved and read.	Your expected result is returned.	The result may differ from expected, depending on the operations performed to the string.
A 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 expected.

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

NOTE

Oversized strings are truncated only if **DBCOMPATIBILITY** is set to **TD** (compatible with Teradata) and **td_compatible_truncation** is set to **on**.

Run the following commands to create **table1** and **table2** to be used in the example:

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

Table 3-3 Example

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
1	SQL_ASCII	UTF8	Yes	<pre>openGauss=# INSERT INTO table1 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78'));</pre>	<pre>id a b c ----+----- +-----+----- 1 87 87 87</pre>	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 incorrect.
2	SQL_ASCII	UTF8	Yes	<pre>openGauss=# INSERT INTO table1 VALUES(2,reverse('123A78'),reverse('123A78'),reverse('123A78')) ;</pre>	<pre>id a b c ----+----- +-----+----- 2 873 873 873</pre>	A string is reversed and then automatically truncated. Therefore, the result is unexpected.
3	SQL_ASCII	UTF8	Yes	<pre>openGauss=# INSERT INTO table1 VALUES(3,'87A123','87A123','87A123');</pre>	<pre>id a b c ----+----- +-----+----- 3 87A1 87A1 87A1</pre>	The column length in the string type is an integer multiple of the length in client character encoding. Therefore, the result is correct after truncation.

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
4	SQL_ASCII	UTF8	No	<pre>openGauss=# INSERT INTO table2 VALUES(1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')); openGauss=# 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 no longer indicate 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 column values can also be scalar expressions.

```
openGauss=# 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:

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

If you do not know some of the column values, you can omit them. In the INSERT statement, if the column name of the specified target table is not displayed, the values to be inserted in the VALUES clause correspond to the columns of the target table based on the column number. That is, the first value in the VALUES clause corresponds to the first column of the target table, the second value of the VALUES clause corresponds to the second column of the target table, and so on. Columns that do not have corresponding values in the VALUES clause are automatically filled with default values or NULL. For example:

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

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

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

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

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

- To insert multiple rows to a table, run the following command:

```
openGauss=# 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:

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

- To delete a backup file, run the following command:

```
openGauss=# DROP TABLE customer_t2 CASCADE;
```

NOTE

If the table to be deleted is dependent on other tables, you need to delete its dependent 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 types of information are required when the **UPDATE** statement is used to update a row:

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

Generally, the SQL language does not provide a unique ID 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 rows by declaring the conditions that must be met by the updated row. 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**:

```
openGauss=# 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. In the statement, **SET** is

followed by the target column and the new column value. The new value can be a constant or an expression.

For example, run the following statement to increase all the values in the **c_customer_sk** column by 100:

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

This statement does not include the **WHERE** clause, so all rows are updated. If the statement includes 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. **WHERE** may not represent an equation and can be replaced by other operators.

You can run an **UPDATE** statement to update multiple columns by specifying multiple values in the **SET** clause. For example:

```
openGauss=# 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 the **VACUUM FULL** statement to maintain 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**:

```
openGauss=# SELECT * FROM pg_tables;
```

- Run the **\d+** command of the **gsq** tool to query table attributes:

```
openGauss=# \d+ customer_t1;
```

- Run the following command to query the data volume of table **customer_t1**:

```
openGauss=# SELECT count(*) FROM customer_t1;
```

- Run the following command to query all data in the table **customer_t1**:

```
openGauss=# SELECT * FROM customer_t1;
```

- Run the following command to query only the data in the column **c_customer_sk**:

```
openGauss=# SELECT c_customer_sk FROM customer_t1;
```

- Run the following command to filter repeated data in the column **c_customer_sk**:

```
openGauss=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
```

- Run the following command to query all data whose column **c_customer_sk** is **3869**:

```
openGauss=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
```

- Run the following command to collate data based on the column **c_customer_sk**:

```
openGauss=# 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.

SQL statements can only access and delete an independent row by declaring conditions that match the row. 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, to delete all the rows whose **c_customer_sk** column is **3869** from the table **customer_t1**, run the following command:

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

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

```
openGauss=# DELETE FROM customer_t1;  
or  
openGauss=# TRUNCATE TABLE customer_t1;
```

NOTE

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

To delete a table, run the following command:

```
openGauss=# DROP TABLE customer_t1;
```

3.2.5 Querying a System Catalog

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.

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 user.

GaussDB provides the following types of system catalogs and views:

- PG-compatible system catalogs and views
These system catalogs and views have the prefix **PG** or **PGXC**.
- New system catalogs and views of GaussDB
These system catalogs and views have the prefix **GS**.
- System catalogs and views that are compatible with Oracle Database
These system catalogs and views have the prefix **ALL**, **DBA**, **USER**, or **PV**.

Querying Database Tables

Create the following tables in the public schema:

```
openGauss=# CREATE TABLE public.search_table_t1(a int) distribute by hash(a);  
CREATE TABLE  
openGauss=# CREATE TABLE public.search_table_t2(b int) distribute by hash(b);  
CREATE TABLE  
openGauss=# CREATE TABLE public.search_table_t3(c int) distribute by hash(c);  
CREATE TABLE
```

```
openGauss=# CREATE TABLE public.search_table_t4(d int) distribute by hash(d);
CREATE TABLE
openGauss=# 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.

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

Information similar to the following is displayed:

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

Viewing Database Users

You can run the **PG_USER** command 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 |      | ***** |      |      | 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 collects the running information about active queries only if the parameter is set to **on**.

Step 2 View the running query statements. Run the following command to 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.

To identify non-idle query statements, run the following command:

```
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 information similar to the following is displayed, the session is successfully terminated:

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

If information similar to the following 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. If the **PG_TERMINATE_BACKEND** function is used to terminate the backend threads of the current session and the user is an initial user, 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. Otherwise, the client fails to be reconnected and the error message "The connection to the server was lost. Attempting reset: Failed." is returned because only the initial user can use password-free login.
2. If the **PG_TERMINATE_BACKEND** function is used to terminate inactive backend threads and the thread pool is opened, idle sessions do not have thread IDs and cannot be ended. In non-thread pool mode, ended 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.
 - Run the following command to create a schema:

```
openGauss=# 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, the object name in the command should consist of the schema name and the object name, which are separated by a dot (.), for example, **myschema.table**.
 - Run the following command to create a schema and specify the owner:

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

If the following information is displayed, the **myschema** schema that belongs to **omm** 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 (.).

 - Run the following command to create the **mytable** table in **myschema**:

```
openGauss=# 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.

- Run the following command to query all data of the **mytable** table in **myschema**:

```
openGauss=# 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.

- Run the following command to view the search path:

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

- Run the following command to set the search path to **myschema,public** (**myschema** will be searched before **public**):

```
openGauss=# 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 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:

- Run the following command to revoke **PUBLIC**'s permission to create objects in the **public** schema. **public** indicates the schema and **PUBLIC** indicates all roles.

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

- Run the following command to view the current schema:

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

- Run the following commands to create user **jack** and grant the **usage** permission for **myschema** to the user:

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

- Run the following command to revoke the usage permission for **myschema** from **jack**:

```
openGauss=# 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 execute the **DROP SCHEMA** command to delete it. For example, run the following command to delete an empty schema named **nullschema**:

```
openGauss=# DROP SCHEMA IF EXISTS nullschema;  
DROP SCHEMA
```
 - To delete a schema that is not null, use the keyword **CASCADE** to delete it and all its objects. For example, run the following command to delete **myschema** and all its objects in it:

```
openGauss=# DROP SCHEMA myschema CASCADE;  
DROP SCHEMA
```
 - Run the following command to delete user **jack**:

```
openGauss=# DROP USER jack;  
DROP ROLE
```

3.2.6.2 Creating and Managing Partitioned Tables

Context

GaussDB supports range partitioned tables.

In a 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.

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.

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

Example 1: using the default tablespace

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

```
openGauss=# 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
```

- Inserting data

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

Suppose a backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can run the following command to insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**:

```

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

```

- Modifying the row movement attributes of the partitioned table

```

openGauss=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE

```

- Deleting a partition

Run the following command to delete partition **P8**:

```

openGauss=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE

```


- Adding a partition

Run the following command to add partition **P8** and set its range to [40000,MAXVALUE]:

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

- Renaming a partition

- Run the following command to rename partition **P8** to **P_9**:

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

- Run the following command to rename partition **P_9** to **P8**:

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

- Querying a partition

Run the following command to query partition **P6**:

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

- Deleting a partitioned table and its tablespaces

```
openGauss=# DROP TABLE tpcds.customer_address;
DROP TABLE
openGauss=# DROP TABLE tpcds.web_returns_p2;
DROP TABLE
```

Example 2: using a user-defined tablespace

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

- Creating tablespaces

```
openGauss=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
openGauss=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
openGauss=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
openGauss=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

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

```
CREATE TABLESPACE
```

- Creating a partitioned table

```
openGauss=# 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
```

- Inserting data

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

Suppose a backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can run the following command to insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**:

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

- Modifying the row movement attributes of the partitioned table

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

- Deleting a partition

Run the following command to delete partition **P8**:

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

- Adding a partition

Run the following command to add partition **P8** and set its range to [40000,MAXVALUE]:

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

- Renaming a partition

- Run the following command to rename partition **P8** to **P_9**:
openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE
- Run the following command to rename partition **P_9** to **P8**:
openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
ALTER TABLE
- Modifying the tablespace of a partition
 - Run the following command to change the tablespace of partition **P6** to **example3**:
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;
ALTER TABLE
 - Run the following command to change the tablespace of partition **P4** to **example4**:
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;
ALTER TABLE
- Querying a partition

Run the following command to query partition **P6**:

```
openGauss=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
openGauss=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
```
- Deleting a partitioned table and its tablespaces

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

3.2.6.3 Creating and Managing Indexes

Background

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 add 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.
- 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

For details about how to create a partitioned table, see [Creating and Managing Partitioned Tables](#).

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

```
openGauss=# 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.

```
openGauss=# 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
```

- Modifying the tablespace of an index partition
 - Change the tablespace of index partition **web_returns_p2_P2_index** to **example1**.

```
openGauss=# 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**.

```
openGauss=# 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
```

- Renaming an index partition

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

```
openGauss=# 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 renaming is successful:

```
ALTER INDEX
```

- Querying indexes

- Run the following command to query all indexes defined by the system and users:

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

- Run the following command to query information about a specified index:

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

- Deleting indexes

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

```
openGauss=# 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:

```
openGauss=# 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.

Run the following command to create an index:

```
openGauss=# 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.

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

- Create a composite index.

Assume you need to frequently query records with **ca_address_sk** being **5050** and **ca_street_number** smaller than **1000** in the **tpcds.customer_address_bak** table. Run the following commands:

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

Run the following command to define a composite index on **ca_address_sk** and **ca_street_number** columns:

```
openGauss=# 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 whose **ca_address_sk** is **5050**, you can create a partial index to facilitate your query.

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

- Create an expression index.

Assume that you need to frequently query records with **ca_street_number** smaller than **1000**, run the following command:

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

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

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

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

```
openGauss=# 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.

A view is different from a base table. It is only a virtual object rather than a physical one. 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. A view is triggered every time it is referenced.

Managing Views

- Creating 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.

```
openGauss=# 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.

- Querying a view

Run the following command to query **MyView**:

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

- Querying views of the current user

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

- Querying all views

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

- Viewing details about a specified view

Run the following command to view details about **MyView**:

```
openGauss=# \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;

- Deleting a view

Run the following command to delete **MyView**:

```
openGauss=# 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.
- Run 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. For example:

```
openGauss=# 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.

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

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

```
CREATE SEQUENCE
```

2. Set the initial value of the **nextval('sequence_name')** function to the default value of a column.

```
openGauss=# 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 the 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 columns or the table where the columns belong.

```
openGauss=# 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:

```
openGauss=# CREATE SEQUENCE newSeq1;
openGauss=# CREATE TABLE newT1
(
    id int not null default nextval('newSeq1'),
    name text
);
openGauss=# 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:

```
openGauss=# INSERT INTO newT1(id, name) SELECT id,name FROM T1;
openGauss=# 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. A large amount of data would be a disaster to the database. Also, the occupied space may dramatically expand. After the import is completed, you need to run **VACUUM FULL** to free space. 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 Oracle. You can create scheduled jobs that are automatically triggered to reduce O&M workload.

This function calls APIs provided by the DBE_TASK package 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).

Managing Scheduled Jobs

Step 1 Create a test table.

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# call dbe_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.
openGauss=# call db_task.submit(2,'call public.prc_job_1();', sysdate, 'interval "1 minute"');

```

-----
(1 row)

```

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

```

openGauss=# 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.

```

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

```

Step 6 Start a job.

```

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

```

Step 7 Modify job attributes.

- Modify the **Next_date** parameter information about a job.
-- Set **Next_date** of **Job1** to 1 hour so that **Job1** will be executed in one hour.

```

openGauss=# 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.
openGauss=# 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**.

```

openGauss=# 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.
openGauss=# call db_task.update(1, 'call public.prc_job_1();', sysdate, 'interval "1 minute"');

```

change
-----

```

```
(1 row)
```

Step 8 Delete a job.

```
openGauss=# call db_task.cancel(1);  
remove
```

```
(1 row)
```

Step 9 View the job execution status.

If a job fails to execute automatically, (the status of **job_status** is **f**), you can query the failure information by visiting the **pg_log** subdirectory of the CN data directory where the job belongs to.

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 who created the job, (**log_user** in **pg_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 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

Introduction

This document describes the database-related design and development specifications in the product design and development process based on the product lifecycle.

This document 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 must be followed during programming and design.
- **Recommendation:** a convention that must be considered 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 in the 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	Distributed table	Data sharding.
	Partitioned table	Data partitioning.
Storage engine	Row store	Row-store table, which is recommended in the point query and a scenario with many add, delete, and alter 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.
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, TIMESTAMPTZ, 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.
Scale-out	Online scale-out	Ensures smooth transition of user services during node scale-out and data redistribution.

- [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

No.	Item	Recommended Value
2	Optimal number of long connections in a cluster	For details, see the default GUC parameter max_connections in the corresponding hardware specifications.
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.

For details about configuring GUC parameters, 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	-

Object Type	Length Restriction (Unit: Byte)	Remarks
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] Do not use a string enclosed in double quotation marks (") to define the database object name, unless you need to specify its capitalization. Case sensitivity of database object names makes fault locating difficult.
- [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 data set.
 - Temporary tables are named in the format of **tmp_Suffix**.

- Unlogged tables are named in the format of **ul_Suffix**.
- Do not create database objects whose names start with **redis_**.
- Do not create database objects whose names start with **mlog_** or **matviewmap_**.

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-, and PG-compatible modes which are compatible with the Teradata syntax, Oracle Database syntax, MySQL syntax, and PostgreSQL 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, a 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 must 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 **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] When deleting a specified database, revoke the CONNECT permission of users on the database to prevent deletion failures caused by active database connections.

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 reasons, see [Database and Schema Design Specifications](#).

4.2.7 Table Design Specifications

- [Rule] View nesting is not recommended.
- [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.
- [Description] Evenly scan each DN when querying tables. Otherwise, DNs 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.

Selecting a Storage Model

[Recommendation] Selecting a storage model is the first step in defining a table. The storage model mainly depends on the customer's service type. For details, see [Table 4-4](#).

Table 4-4 Table storage models and scenarios

Storage Model	Application Scenario
Row store	- Point queries (simple index-based queries that only return a few records) - Scenarios requiring frequent addition, deletion, and modification

Selecting a Distribution Mode

Comply with the following rules to distribute table data.

Table 4-5 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenario
Hash	Table data is distributed on all DN's 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 DN's.	Users need to customize distribution rules.
List	Table data is mapped to specified columns based on specific values and distributed to corresponding DN's.	Users need to customize distribution rules.

NOTE

When hash, range, or list distribution is specified, the primary key and unique index to be created must contain distribution keys.

- [Description] Evenly distribute table data on each DN to prevent data skew. If most data is stored on several DN's, 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, the replication distribution policy is recommended.

```
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.
 - d. According to the database specifications, the primary key of the hash distributed table must contain the distribution key. Therefore, you can select the primary key of the table as the distribution key.

Table 4-6 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

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] Set the upper limit of a partition to **MAXVALUE** to prevent data overflow.

The example of defining a partitioned table is as follows:

```
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)
```

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 for 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>), 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
	BLOB (binary large object) and RAW (variable-length hexadecimal string)	Not recommended

Data Type	Description	Recommended or Not
Bit string	BIT(<i>n</i>) and VARBIT(<i>n</i>)	Recommended
Special character	NAME and "CHAR" are usually used within the database system.	Not recommended
JSON	JSON data does not support operators.	Not recommended
User-defined	Defines enumerated types.	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 number, 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 oversized fields (such as varchar(1000) and varchar(4000)) cannot exceed 8.
- [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] 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] 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.

- The data type is BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, or INTERVAL.
- The length of the default value cannot exceed 128 bytes.
- The default value of the added column does not contain the volatile function.
- 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

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

- [Specification] For a distributed hash table, the primary key and unique index must contain distribution keys.
- [Rule] Properly design composite indexes to avoid redundancy.
For example, if an index has been created for **(a, b, c)**, you shall not create an index for **(a)**, **(b)**, **(c)**, **(a, b)**, or **(b, c)** independently. If only the filtering condition of the **a** column is used for a query, composite indexes are used for the query as well.
- [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 and composite indexes) columns must 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**.

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

- [Description] Row-store tables support UNIQUE constraints.
- [Recommendation] The constraint name should indicate that it is a unique constraint, for example, **UNI***Included columns*.

Primary Key Constraints

- [Description] Row-store tables support PRIMARY KEY constraints.
- [Recommendation] The constraint name should indicate that it is a primary key constraint, for example, **PK***Column name*.

Check Constraints

- [Description] Row-store tables support 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

- [Proposal] Do not nest views unless they have strong dependency on each other.
- [Proposal] Try to avoid collation operations in a view definition.

Joined Table Design

- [Proposal] Minimize joined columns across tables.

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

4.3 Database Design 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 the same table columns in WHERE conditions.

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 code does not meet specifications:

```
-- The phonenum 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 phonenum = 13512345678;
```

- The following modification should be considered:

```
-- The phonenum 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 phonenum = '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, leading to unnecessary performance overhead.

Example:

- The following lines of code 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 instead).
- [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 DN 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 reltuples 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] LIMIT cannot be directly used in the UPDATE statement. The WHERE condition must be used to specify the target row to be updated.
- [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.
- [Specification] Multi-table update is not supported.
Multi-table update is to update multiple tables in a single SQL statement.
- [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] LIMIT cannot be used in the DELETE statement. The WHERE condition should be used to specify the target row to be deleted.
- [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.
- [Specification] Multi-table deletion is not supported.
Multi-table deletion indicates that multiple tables are deleted in a single SQL statement.
- [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 cannot 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;
```

Run the following command before the transaction ends:

```
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.
- [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.

When multiple statements are combined into one statement that contains object operations, if the intermediate object operation fails, a new transaction is started to execute subsequent statements.

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();
}
```

In the preceding statement, if

```
CREATE TABLE t1;
```

The creation fails, and a new transaction will be started.


```
DROP TABLE t1;
```

The execution fails.

- 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

- [Recommendation] In GaussDB, you are advised to perform DDL operations (such as CREATE TABLE and COMMENT) in a unified manner. Avoid DDL operations in batch processing to avoid performance deterioration caused by many concurrent transactions.
- [Recommendation] Perform the TRUNCATE operation immediately after an unlogged table is used, because GaussDB cannot ensure the security of unlogged tables in abnormal scenarios.
- [Recommendation] Suggestions on the storage model of temporary and unlogged tables are the same as those on base tables. Create temporary tables in the same storage model as the base table to avoid high computing costs caused by hybrid join.
- [Recommendation] 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.
- [Recommendation] 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

- [Recommendation] Explicitly provide the inserted column list in the INSERT statement. Example:

```
INSERT INTO task(name,id,comment) VALUES ('task1','100','100th task');
```
- [Recommendation] 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.
- [Recommendation] 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

- [Recommendation] Convert data types explicitly. If you perform implicit conversion, the result may differ from expected.
- [Recommendation] During data query, explicitly specify the data type for constants, and do not attempt to perform any implicit data type conversion.
- [Description] 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

- [Recommendation] 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.
- [Recommendation] Perform DDL and DML operations encapsulated in transactions. For example, operations such as TRUNCATE TABLE, UPDATE TABLE, DELETE TABLE, and DROP TABLE cannot be revoked once they are committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- [Recommendation] During query compilation, you are advised to list all columns to be queried and avoid using **SELECT ***. Doing so reduces output lines, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- [Recommendation] During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- [Recommendation] The cost of joining more than three tables or views, especially FULL JOIN, 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.
- [Recommendation] Avoid using a Cartesian product or FULL JOIN. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.
- [Description] 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.
- [Description] 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.
- [Description] 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.
- [Description] 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.

- [Description] Null records are not counted when count(distinct col) is used to calculate the number of non-NULL columns that are not repeated.
- [Description] 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.
- [Recommendation] 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.
- [Recommendation] Use the following time-related macros 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 deteriorates severely.

Table 4-10 Time-related macros

Macro Name	Description	Example
CURRENT_DATE	Obtains the current date, excluding the hour, minute, and second details.	<pre>openGauss=# SELECT CURRENT_DATE; date ----- 2018-02-02 (1 row)</pre>
CURRENT_TIME	Obtains the current time, excluding the year, month, and day.	<pre>openGauss=# SELECT CURRENT_TIME; timetz ----- 00:39:34.633938+08 (1 row)</pre>
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.	<pre>openGauss=# SELECT CURRENT_TIMESTAMP(6); timestampz ----- 2018-02-02 00:39:55.231689+08 (1 row)</pre>

- [Recommendation] Do not use scalar subquery statements. A scalar subquery appears in the output list of a SELECT statement. In the following example, the underlined part 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 joins based on the service logic.
- [Recommendation] 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.

- [Recommendation] 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. For 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;
```
- [Recommendation] 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.
- [Recommendation] 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.
- [Recommendation] Do not rely on only the LIMIT clause to return the result set displayed in a specific sequence. Combine ORDER BY and LIMIT clauses for some specific result sets and use OFFSET to skip specific results if necessary.
- [Recommendation] If the service logic is accurate, you are advised to use UNION ALL instead of UNION.
- [Recommendation] 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:

```
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)
```
- [Recommendation] 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.
- [Recommendation] Replace (NOT) IN with (NOT) EXIST when joined columns do not contain NULL values. For example, in the following 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.
- [Recommendation] 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.

4.4 Client Programming Specifications

4.4.1 JDBC

- [Description] When a third-party tool connects to GaussDB through JDBC, JDBC sends a connection request to GaussDB. By default, the following parameters are added. For details, see the implementation of the ConnectionFactoryImpl 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.

In precision-sensitive scenarios, the numeric type is recommended.

- [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 bytes. 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] 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**.

5 Application Development Guide

5.1 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 5-1 describes the compatibility of application development drivers.

Table 5-1 Compatibility

Driver	Compatibility
JDBC, ODBC, libpq, and Psycopg	The drivers are compatible with the earlier database version. However, to use the new features added to the drivers and database, you must upgrade the database.

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

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

5.2 Development Based on JDBC

Java Database Connectivity (JDBC) is Java API for executing SQL statements, providing unified accessing API for different relational databases, based on which applications process data. GaussDB supports JDBC 4.0 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridge.

5.2.1 JDBC Package, Driver Class, and Environment Class

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-2](#).

Table 5-2 Driver package download list

Version	Download Address
V2.0-2.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 5-2](#).
 - 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:

- **gsjdbc4.jar**: This driver package applies to the scenario where services are migrated from PG. 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.
- **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. This driver package is used when both PG and GaussDB are accessed in a JVM process.

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.
 - **gsjdbc200.jar**: This driver package applies to the scenario where services are migrated from Gauss200. 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.
 - 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 the GaussDB database. 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.
-

Driver Class

Before creating a database connection, load the database driver class **org.postgresql.Driver** (decompressed from **gsjdbc4.jar**).

 NOTE

1. 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.
2. JDBC of this version does not support identity & access management suite (IAM) for authentication.
3. Compared with the PG driver, the GaussDB JDBC driver has the following enhanced features:
 1. SHA-256 is supported for encrypted login.
 2. The third-party log framework that implements the sf4j API can be connected.
 3. Distributed load balancing at the connection level is supported.
 4. DR failover is supported.

Environment Class

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 how to configure JDK.

- Step 1** Enter **java -version** in the MS-DOS window (command prompt in Windows) to check the JDK version. Ensure that the JDK version is JDK 1.8. If the JDK is not installed, download the installation package and install it.
- Step 2** On the Windows desktop, right-click **This PC** and choose **Properties** from the shortcut menu.
- Step 3** In the displayed **System** window, click **Advanced system settings** in the navigation tree on the left.
- Step 4** In the **System Properties** dialog box, click **Environment Variables** in the lower right corner.
- Step 5** In the **System variables** area of the **Environment Variables** dialog box that is displayed, set the following variables.

Variable	Operation	Variable Value
JAVA_HOME	• 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	Edit	• If JAVA_HOME is configured, add %JAVA_HOME%\bin before the variable value. • If JAVA_HOME is not configured, add the full Java installation path before the variable value: C:\Program Files\Java\jdk1.8.0_131\bin;
CLASSPATH	New	.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;

Step 6 Click **OK** and close the windows one by one.

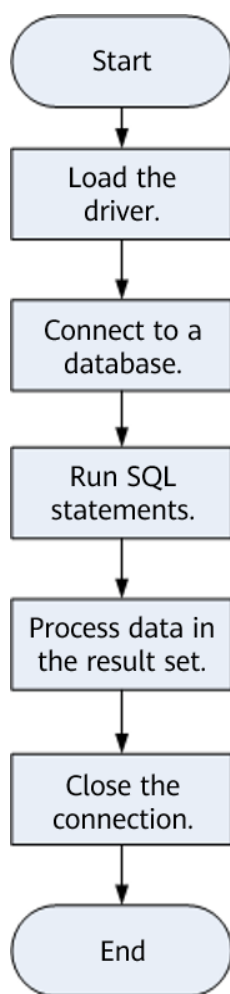
----End

5.2.2 Development Process

Figure 5-1 shows the JDBC-based development process.

This process involves the following steps: loading the driver, connecting to a database, executing SQL statements, processing the result set, and closing the connection.

Figure 5-1 Application development process based on JDBC



5.2.3 Loading a Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Before creating a connection, implicitly load the driver in the code using the `Class.forName("org.postgresql.Driver")` method.
- During the JVM startup, transfer the driver as a parameter to JVM using the `java -Djdbc.drivers=org.postgresql.Driver jdbctest` argument.

NOTE

- **jdbctest** is the name of a test application.
- If **opengaussjdbc.jar** is used, change the driver class name to **com.huawei.opengauss.jdbc.Driver**.

5.2.4 Connecting to a Database

After a database is connected, you can use JDBC to run SQL statements to operate data.

Function Prototype

JDBC provides the following three database connection methods:

- `DriverManager.getConnection(String url);`
- `DriverManager.getConnection(String url, Properties info);`
- `DriverManager.getConnection(String url, String user, String password);`

Parameters

Table 5-3 Database connection parameters

Parameter	Description
url	gsjdbc4.jar database connection descriptor. The format is as follows: • jdbc:postgresql:database • jdbc:postgresql://host/database (If the port number is not specified, the default port number is used.) • jdbc:postgresql://host:port/database • jdbc:postgresql://host:port/database?param1=value1&param2=value2 • jdbc:postgresql://host1:port1,host2:port2/database?param1=value1&param2=value2 NOTE If gsjdbc200.jar is used, replace jdbc:postgresql with jdbc:gaussdb. • database indicates the name of the database to connect. • host indicates the name or IP address of the database server. For security purposes, the database CN forbids access from other nodes in the cluster without authentication. 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. If it is deployed inside, 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 property. The parameter can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign a value to the parameter, and uses an ampersand (&) to separate parameters. You can also use the attributes of the info object for configuration. For details, see the example below. • value indicates the corresponding database connection property value. • 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. • In a distributed environment, you are advised to configure the autoBalance parameter for the connection strings for load balancing and configure at least two CNs to prevent connection setup failures due to node faults.

Parameter	Description
info	Database connection properties (all properties are case-sensitive). Common properties are described as follows: • PGDBNAME: string type. This parameter specifies the database name. You do not need to set this parameter in the URL because the database name is automatically parsed from the URL. • PGHOST: string type. This parameter specifies the host IP address. Use colons (:) to separate IP addresses and port numbers, and use commas (,) to separate multiple CNs. (This parameter does not need to be set in the URL. The system automatically parses the URL to obtain its value.) • PGPORT: integer type. This parameter specifies the host port number. Use colons (:) to separate IP addresses and port numbers, and use commas (,) to separate multiple CNs. (This parameter does not need to be set in the URL. The system automatically parses the URL to obtain its value.) • user: string type. This parameter specifies the database user who creates the connection. • password: string type. This parameter specifies the password of the database user. • loggerLevel: string type. The default value is NULL, which is equivalent to INFO. The following log levels are supported: OFF, INFO, DEBUG, and TRACE. Set this parameter to OFF to disable the log function. INFO, DEBUG, and TRACE logs record information of different levels. • loggerFile: string type. This parameter specifies the log output path (directory and file name). 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. • logger: string type. It indicates the log output framework used by the JDBC driver. The JDBC driver supports the log output framework used for interconnecting with applications. Currently, only the Slf4j-API-based third-party log output framework is supported. For details, see 6.2.9 Log Management. 1. If it is left empty or is set to JDK LOGGER, JDK LOGGER is used. 2. Otherwise, the slf4j-API-based third-party log framework must be used. • allowEncodingChanges: Boolean type. The default value is false. If this parameter is set to true, the character set type can be changed. This parameter is used together with characterEncoding=CHARSET to set the character set. The two parameters are separated by ampersand (&). The value of characterEncoding can be UTF8, GBK, LATIN1, or GB18030.

Parameter	Description
	currentSchema: string type. This parameter specifies the schema to be set in search-path. If the schema name contains special characters except letters, digits, and underscores (<code>_</code>), you are advised 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 (<code>,</code>). Schemas containing special characters also need to be enclosed in quotation marks. For example, currentSchema=<code>schema_a,"schema-b","schema/c"</code>. loadBalanceHosts: Boolean type. In the default mode (disabled), multiple hosts specified in the URL are connected in sequence. If load balancing is enabled, the shuffle algorithm is used to randomly select a host from the candidate hosts to establish a connection. autoBalance: string type. - 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. Example: <code>jdbc:postgresql://host1:port1,host2:port2/database?autoBalance=true</code> JDBC 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 refreshCNIPListTime parameter specifies the interval for obtaining the list, and the default value is 10s. Hosts obtained from the CN list are data IP addresses. 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 new connections will be established on host1, host2, host3, and host4 using the RoundRobin algorithm. priorityn indicates that the JDBC-based load balancing function is enabled. Multiple connections of an application are balanced to the first n available CNs configured in the URL. When the first n CNs are unavailable, connections are randomly allocated to other available CNs in the database cluster. n is a number not less than 0 and less than the number of CNs configured in the URL. Example: <code>jdbc:postgresql://host1:port1,host2:port2,host3:port3,host4:port4/database?autoBalance=priority2</code> JDBC periodically obtains the list of available CNs in the entire cluster (defined by refreshCNIPListTime). 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
	The JDBC driver preferentially selects host1 and host2 for load balancing. If both host1 and host2 are unavailable, the JDBC driver randomly selects a CN from host3, host4, host5, and host6 for connection. 3. 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:postgresql://host1:port1,host2:port2,host3:port3/database?autoBalance=shuffle JDBC 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 refreshCNIPListTime parameter specifies the interval for obtaining the list, and the default value is 10s. 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. 4. If this parameter is set to false, the JDBC load balancing and priority-based load balancing functions are disabled. The default value is false. CAUTION 1. 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. 2. Load balancing can be used only in distributed scenarios. 3. 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. • refreshCNIPListTime: integer type. It obtains the cache validity period of the CNs. During connection establishment, the CN status in the database cluster is checked. The status is trusted within the period specified by refreshCNIPListTime. After the period expires, the status becomes invalid. The IP address list of available CNs is obtained again when the next connection is established. The default value is 10 seconds. • hostRecheckSeconds: integer type. After JDBC attempts to connect to a host, the host status is saved: connection success or connection failure. This status is trusted within the duration specified by hostRecheckSeconds. After the duration expires, the status becomes invalid. The default value is 10 seconds. • ssl: Boolean type. This parameter specifies a connection in SSL mode.

Parameter	Description
	When ssl is set to true, the NonValidatingFactory channel and certificate mode are supported. 1. For the NonValidatingFactory channel, configure the username and password and set SSL to true. 2. In certification mode, configure the client certificate, key, and root certificate, and set SSL to true. • sslmode: string type. This parameter specifies the SSL authentication mode. The value can be disable, allow, prefer, require, verify-ca, or verify-full. – disable: SSL connection is not enabled. – 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: 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. • sslcert: string type. This parameter specifies the complete path of the certificate file. The type of the client and server certificates is End Entity. • sslkey: string type. This parameter specifies the complete path of the key file. You need to convert the client certificate to the DER format. For details, see Connecting to the Database (Using SSL). • sslrootcert: string type. This parameter specifies the name of the SSL root certificate. The root certificate type is CA. • sslpassword: string type. This parameter is provided for ConsoleCallbackHandler. • sslpasswordcallback: string type. This parameter specifies the class name of the SSL password provider. The default value is org.postgresql.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler. • sslfactory: string type. This parameter specifies the class name used by SSLSocketFactory to establish an SSL connection. • sslprivatekeyfactory: string type. This parameter specifies the fully qualified name of the implementation class of the

Parameter	Description
	org.postgresql.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 org.postgresql.ssl.BouncyCastlePrivateKeyFactory. You need to provide the bcpkix-jdk15on.jar package. The recommended version is 1.65 or later. • sslfactoryarg: string type. The value is an optional parameter of the constructor function of the sslfactory (This parameter is not recommended). • sslhostnamerverifier: string type. This parameter 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. • loginTimeout: integer type. This parameter specifies the waiting time for establishing the database connection, in seconds. 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. • connectTimeout: integer type. It 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. If the value is 0 (unit: second), the timeout mechanism is disabled. When multiple IP addresses are configured in the URL, this parameter indicates the timeout interval for connecting to a single IP address. • socketTimeout: integer type. It specifies 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. If the value is 0 (unit: second), the timeout mechanism is disabled. When the timeout is triggered on the JDBC side and the connection is closed, the running services delivered by the JDBC to the database are forcibly terminated. This capability is controlled by the GUC parameter check_disconnect_query. If this parameter is set to on, the capability is supported. If this parameter is set to off, the capability is not supported. • socketTimeoutInConnecting: integer type. This parameter specifies the timeout value of the socket read operation during the connection establishment. If the time taken to read data from the server exceeds the value of this parameter, JDBC searches for the next node for connection. The unit of the timeout interval is second. The default value is 5s. • driverInfoMode: string type. This parameter controls the output mode of the driver description information. The value can be postgresql or gaussdb. The default value is postgresql, indicating that the driver description related to PG is displayed. If this parameter is set to gaussdb, the driver description related to GaussDB is displayed.

Parameter	Description
	• cancelSignalTimeout: integer type. It 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. The default value is 10 seconds. • tcpKeepAlive: Boolean type. This parameter is used to enable or disable TCP keepalive detection. The default value is false. • logUnclosedConnections: Boolean type. The default value is false. The client may leak a connection object because it does not call the connection object's close() method. These objects will be collected as garbage and finalized using the finalize() method. After this parameter is set to true, if the caller ignores this operation, this method closes the connection. • assumeMinServerVersion (deprecated): string type. This parameter indicates the version of the server to connect. • ApplicationName: string type. This parameter specifies the name of the application that is being connected. You can query the pgxc_stat_activity table on the CN to view information about the client that is being connected. The name is displayed in the application_name column. The default value is PostgreSQL JDBC Driver. • connectionExtraInfo: Boolean type. This parameter specifies whether the JDBC driver reports the driver deployment path and process owner to the database. The value can be true or false. The default value is false. If connectionExtraInfo is set to true, the JDBC driver reports the driver deployment path and process owner 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. • autosave: string type. The value can be always, never, or conservative. The default value is never. This parameter specifies the action that the driver should perform upon a query failure. If autosave is set to always, the JDBC driver sets a savepoint before each query and rolls back to the savepoint if the query fails. If autosave is set to never, there is no savepoint. If autosave is set to conservative, a savepoint is set for each query. However, the system rolls back and retries only when there is an invalid statement. • protocolVersion: integer type. This parameter specifies the connection protocol version. Only versions 1 and 3 are supported. Note: If this parameter is set to 1, only the V1 server is connected. MD5 encryption is used when this parameter is set to 3. You must need to set the GUC parameter password_encryption_type to 1 to change the database encryption type. After the database is restarted, create a user who uses MD5 encryption to encrypt passwords. You must also change the client

Parameter	Description
	connection mode to md5 in the pg_hba.conf file. Log in to the system as the new user. (You are advised not to set this parameter because the MD5 encryption algorithm has lower security and poses security risks.) NOTE The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm. • prepareThreshold: integer type. This parameter specifies the number of times that the PreparedStatement object is executed before the prepared statement on the server is used. The default value is 5, indicating 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: integer type. This parameter specifies the maximum number of queries generated by the cache statement object of each connection. The default value is 256. If the number of queries generated by the statement object is greater than 256, the least recently used queries will be discarded from the cache. The value 0 indicates that the cache function is disabled. • preparedStatementCacheSizeMiB: integer type. This parameter specifies the maximum number of queries generated by the cache statement object of each connection. The unit is MB. The default value is 5. If the size of the cached queries exceeds 5 MB, the least recently used query cache will be discarded. The value 0 indicates that the cache function is disabled. • databaseMetadataCacheFields: integer type. The default value is 65536. This parameter specifies the maximum number of columns that can be cached in each connection. The value 0 indicates that the cache function is disabled. • databaseMetadataCacheFieldsMiB: integer type. The default value is 5. This parameter indicates the maximum size of columns that can be cached in each connection, in MB. The value 0 indicates that the cache function is disabled. • stringtype: string type. The value can be unspecified or varchar. This parameter specifies the type of the PreparedStatement parameter used by the setString() method. If stringtype is set to varchar, these parameters are sent to the server as varchar parameters. If stringtype is set to unspecified, these parameters are sent to the server as an untyped value, and the server attempts to infer their appropriate type. • batchMode: string type. This parameter specifies whether to connect the database in batch mode. The default value is on, indicating that the batch mode is enabled. 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. 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.

Parameter	Description
	• fetchsize: integer type. This parameter specifies the default fetchsize for statements in the created connection. The default value is 0, indicating that all results are obtained at a time. It has the same function as defaultRowFetchSize. If they are set at the same time, fetchsize is used. • reWriteBatchedInserts: Boolean type. During batch import, set this parameter to true to combine <i>N</i> insertion statements into one: insert into TABLE_NAME values(values1, ..., valuesN), ..., (values1, ..., valuesN). To use this parameter, set batchMode to off. • unknownLength: integer type. The default value is Integer.MAX_VALUE. This parameter specifies the length of the unknown length type when the data of some PG types (such as TEXT) is returned by functions such as <code>ResultSetMetaData.getColumnDisplaySize</code> and <code>ResultSetMetaData.getPrecision</code>. • defaultRowFetchSize: integer type. This parameter specifies the number of rows read by fetch in <code>ResultSet</code> at a time. Limiting the number of rows read each time in a database access request can avoid unnecessary memory consumption, thereby avoiding out of memory exception. The default value is 0, indicating that all rows are obtained at a time in <code>ResultSet</code>. This parameter cannot be set to a negative value. • binaryTransfer: Boolean type. This parameter specifies whether data is sent and received in binary format. The default value is false. • binaryTransferEnable: string type. This parameter specifies the type for which binary transmission is enabled. Every two types are separated by commas (,). You can select either the OID or name, for example, binaryTransferEnable=INT4_ARRAY,INT8_ARRAY. For example, if the OID name is BLOB and the OID number is 88, you can configure the OID as follows: binaryTransferEnable=BLOB or binaryTransferEnable=88 • binaryTransferDisable: string type. This parameter specifies the type for which binary transmission is disabled. Every two types are separated by commas (,). You can select either the OID or name. The value of this parameter overwrites the value of binaryTransferEnable. • blobMode: string type. This parameter is used to set the data type of parameters bound to the <code>setBinaryStream</code> method. If the value is on, the data type is blob. If the value is off, the data type is bytea. The default value is on. You are advised to set this parameter to on for systems migrated from Oracle Database and MySQL and to off for systems migrated from PG. • socketFactory: string type. This parameter specifies the name of the class used to create a socket connection with the server. This class must implement the <code>javax.net.SocketFactory</code> API and define a constructor with no parameter or a single string parameter.

Parameter	Description
	• socketFactoryArg: string type. The value is an optional parameter of the constructor function of the socketFactory class and is not recommended. • receiveBufferSize: integer type. This parameter is used to set SO_RCVBUF on the connection stream. • sendBufferSize: integer type. This parameter is used to set SO_SNDBUF on the connection stream. • preferQueryMode: string type. The value can be extended, extendedForPrepared, extendedCacheEverything, or simple. This parameter specifies the query mode. The default value is extended. In simple mode, only the Q message in text mode can be sent. The parse and bind messages are not supported. In extended mode, parse, bind, and execute messages are used. In extendedForPrepared mode, only the prepared statement object uses extended query, and the statement object uses only simple query. The extendedCacheEverything mode caches the query generated by each statement object. • ApplicationType: string type. The value can be not_perfect_sharding_type or perfect_sharding_type. It specifies whether to enable distributed write and query. The default value is not_perfect_sharding_type. Distributed write and query are enabled if this parameter is set to not_perfect_sharding_type. If it is set to perfect_sharding_type, distributed write and query are disabled by default. Distributed write and query can be performed only when /* multinode */ is added to the SQL statement. This parameter is valid only when the database is in the GTM-free scenario. • priorityServers: integer type. This value is used to specify the first <i>n</i> nodes configured in the URL as the primary cluster to be connected preferentially. The default value is null. The value is a number greater than 0 and less than the number of CNs configured in the URL. It is used in streaming DR scenarios. 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 DR cluster nodes. • usingEip: Boolean type. The value specifies whether to use the 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. • traceInterfaceClass: string type. The default value is null, which is used to obtain the implementation class of traceld. The value is the fully qualified name of the implementation class of the org.postgresql.log.Tracer API that implements the method for obtaining traceld. • use_boolean: Boolean type. This parameter is used to set the OID type bound to the setBoolean method in extended mode. The default value

Parameter	Description
	is false, indicating that the int2 type is bound. The value true indicates that the Boolean type is bound. • allowReadOnly: Boolean type. This parameter specifies whether the read-only mode is allowed. The default value is true, indicating that the read-only mode is allowed. If this parameter is set to false, the read-only mode is disabled. • TLSCiphersSupported: string type. This parameter is used to set the supported TLS encryption suite. The default value is 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: Boolean type. The default value is false. If this parameter is set to true, trailing 0s of the numeric type are removed. This parameter is valid only for ResultSet.getObject(int columnIndex). • enableTimeZone: Boolean type. The default value is true. This parameter specifies whether to enable the time zone setting on the server. The value true indicates that the JVM time zone is obtained to specify the database time zone. The value false indicates that the database time zone is used. • socketTimeoutInConnecting: integer type. The default value is 5s. It specifies the timeout value for a socket read operation during connection establishment. If the time taken to read data from a server exceeds the value specified during connection establishment, the connection is closed. If the value is 0, the timeout mechanism is disabled. • driverInfoMode: string type. This parameter controls the output mode of the driver description information. The value can be postgresql or gaussdb. The default value is postgresql, indicating that the driver description related to PG is displayed. If this parameter is set to gaussdb, the driver description related to GaussDB is displayed.
user	Database user.
password	Password of the database user.

Examples

```
// The following uses gsjdbc4.jar as an example.  
// The following code encapsulates database connection operations into an API. The database can then be
```

```
connected using an authorized username and a password.
public static Connection getConnect(String username, String passwd)
{
    // Driver class.
    String driver = "org.postgresql.Driver";
    // Database connection descriptor.
    String sourceURL = "jdbc:postgresql://$ip:$port/postgres";
    Connection conn = null;

    try
    {
        // Load the driver.
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return 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;
}

// The following code uses the Properties object as a parameter to establish a connection.
public static Connection getConnectUseProp(String username, String passwd)
{
    // Driver class.
    String driver = "org.postgresql.Driver";
    // Database connection descriptor.
    String sourceURL = "jdbc:postgresql://$ip:$port/postgres?autoBalance=true";
    Connection conn = null;
    Properties info = new Properties();

    try
    {
        // Load the driver.
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return null;
    }

    try
    {
        info.setProperty("user", userName);
        info.setProperty("password", password);
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL, info);
        System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }
}
```

```
    return conn;
}
```

5.2.5 Connecting to the Database (Using SSL)

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.

This section describes how applications configure the client in SSL mode through JDBC. Contact an administrator to configure the server.

To use the method described in this section, you must have the server certificate, client certificate, and private key files. For details how to obtain these files, see related documents and commands of OpenSSL.

Configuring the Client

Upload the certificate files **client.key.pk8**, **client.crt**, and **cacert.pem** generated in "Configuring the Server" to the client. The path is specified in the code.

Examples

Note: Select either example 1 or example 2.

```
// 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).
public class SSL{
    public static void main(String[] args) {
        Properties urlProps = new Properties();
        String url = "jdbc:postgresql://$ip:$port/postgres";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");

        /**
         * ===== Example 1: Use the NonValidatingFactory channel.
         */
        urlProps.setProperty("sslfactory", "org.postgresql.ssl.NonValidatingFactory");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("password", password);
        urlProps.setProperty("ssl", "true");
        /**
         * ===== Examples 2: Use a certificate.
         */
        urlProps.setProperty("sslcert", "client.crt");
        urlProps.setProperty("sslkey", "client.key.pk8");
        urlProps.setProperty("sslrootcert", "cacert.pem");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("ssl", "true");
        /* sslmode can be set to require, verify-ca, or verify-full. Select one from the following three
        examples.*/
        /* ===== Example 2.1: Set sslmode to require to use the certificate for authentication.
        */
        urlProps.setProperty("sslmode", "require");
        /* ===== Example 2.2: Set sslmode to verify-ca to use the certificate for
        authentication. */
        urlProps.setProperty("sslmode", "verify-ca");
        /* ===== Example 2.3: Set sslmode to verify-full to use the certificate (in the Linux
        OS) for authentication. */
```

```
urls = "jdbc:postgresql://world:8000/postgres";
urlProps.setProperty("sslmode", "verify-full");

try {
    Class.forName("org.postgresql.Driver").newInstance();
} catch (Exception e) {
    e.printStackTrace();
}

try {
    Connection conn;
    conn = DriverManager.getConnection(urls,urlProps);
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}

}
}

/**
 * Note: Convert the client key to the DER format.
 * 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 due to their low security.
 * 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 package for projects that use JDBC. The
 * recommended version is 1.65 or later.
 */
```

5.2.6 Connecting to a Database (Using UDS)

The UDS is used for data exchange between different processes on the same host. You can add **unixsocket** to obtain the socket factory.

The **unixsocket-core-XXX.jar**, **unixsocket-common-XXX.jar**, and **unixsocket-native-common-XXX.jar** JAR packages need to be referenced. In addition, you need to add **socketFactory=org.newsclub.net.unix.AFUNIXSocketFactory** **\$FactoryArg&socketFactoryArg=** *[path-to-the-unix-socket]* to the URL connection string.

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).
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Properties;

public class Test {
    public static void main(String[] args) {
        String driver = "org.postgresql.Driver";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Connection conn;
        try {
            Class.forName(driver).newInstance();
```

```
Properties properties = new Properties();
properties.setProperty("user", userName);
properties.setProperty("password", password);
conn = DriverManager.getConnection("jdbc:postgresql://$ip:$port/postgres?
socketFactory=org.newsclub +
    ".net.unix" +
    ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=/data/tmp/s.PGSQL.8000",
    properties);
System.out.println("Connection Successful!");
Statement statement = conn.createStatement();
statement.executeQuery("select 1");
} catch (Exception e) {
    e.printStackTrace();
}
}
```

NOTICE

- Set the **socketFactoryArg** parameter based on the actual path. The value must be the same as that of the GUC parameter **unix_socket_directory**.
- The connection host name must be set to **localhost**.

5.2.7 Running SQL Statements

Running a Common SQL Statement

To enable an application to operate data in the database by running SQL statements (statements that do not need to transfer parameters), perform the following operations:

Step 1 Create a statement object by calling the **createStatement** method in **Connection**.

```
// 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).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
Statement stmt = conn.createStatement();
```

Step 2 Run the SQL statement by calling the **executeUpdate** method in **Statement**.

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));");
```

 NOTE

- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. The VACUUM operation 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.
- When **prepareThreshold** is set to **1**, each SQL statement executed by the statement is cached because cached statements are not evicted by default (default value of **preferQueryMode**). As a result, memory bloat may occur. In this case, set **preferQueryMode** to **extendedCacheEverything** to evict cached statements.

Step 3 Close the statement object.

```
stmt.close();
```

----End

Running a Prepared SQL Statement

Prepared statements are compiled and optimized once but can be used in different scenarios by assigning multiple values. Using prepared statements improves execution efficiency. If you want to run a statement for several times, use a prepared statement. Perform the following operations:

Step 1 Create a prepared statement object by calling the **prepareStatement** method in **Connection**.

```
PreparedStatement pstmt = con.prepareStatement("UPDATE customer_t1 SET c_customer_name = ?  
WHERE c_customer_sk = 1");
```

Step 2 Set parameters by calling the **setShort** method in **PreparedStatement**.

```
pstmt.setShort(1, (short)2);
```

 CAUTION

After binding parameters are set in **PreparedStatement**, 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, you need to evaluate and control the size of the bound data to prevent the size from exceeding the upper packet limit.

Step 3 Run the prepared statement by calling the **executeUpdate** method in **PreparedStatement**.

```
int rowcount = pstmt.executeUpdate();
```

Step 4 Close the prepared statement object by calling the **close** method in **PreparedStatement**.

```
pstmt.close();
```

----End

Calling a Stored Procedure

To call an existing stored procedure by using JDBC in GaussDB, perform the following operations:

Step 1 Create a call statement object by calling the **prepareCall** method in **Connection**.

```
// 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).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection myConn = DriverManager.getConnection("url",userName,password);
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

Step 2 Set parameters by calling the **setInt** method in **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 Call the stored procedure by calling the **execute** method in **CallableStatement**.

```
cstmt.execute();
```

Step 5 Obtain the output parameter by calling the **getInt** method in **CallableStatement**.

```
int out = cstmt.getInt(4); // Obtain the OUT parameter.
```

Example:

```
// The following stored procedure (containing the OUT parameter) has been created:
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;
/
```

Step 6 Close the call statement by calling the **close** method in **CallableStatement**.

```
cstmt.close();
```

 NOTE

- Many database classes such as Connection, Statement, and ResultSet have a close() method. Close these classes after using their objects. Closing Connection will close all the related Statements, and closing a Statement will close its ResultSet.
- 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 you bind parameters in `myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}")` during a stored procedure calling, you can bind parameters according to the placeholders sequence and register the first or the fourth parameter as the output parameter. The preceding example registers the fourth parameter.

NOTICE

- 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 an SQL statement must be run separately.
 - Output parameters must be registered for parameters of the inout type in the stored procedure.
-

----End

Batch Processing

When a prepared statement processes multiple pieces of similar data, the database creates only one execution plan. This improves compilation and optimization efficiency. Perform the following operations:

Step 1 Create a prepared statement object by calling the **prepareStatement** method in **Connection**.

```
// 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).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

Step 2 Call the **setShort** parameter for each piece of data, and call **addBatch** to confirm that the setting is complete.

```
pstmt.setShort(1, (short)2);
pstmt.addBatch();
```

Step 3 Perform batch processing by calling the **executeBatch** method in **PreparedStatement**.

```
int[] rowcount = pstmt.executeBatch();
```


Step 4 Close the prepared statement object by calling the **close** method in **PreparedStatement**.

```
pstmt.close();
```

 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 rows at a time. The statement for disabling automatic commit is **conn.setAutoCommit(false);**.

----End

Adding Single-Shard Execution Syntaxes to Statements

Step 1 Set the **nodeName** parameter by calling **setClientInfo("nodeName","dnx")** in **Connection**.

```
Connection conn = getConnection();  
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

1. 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.
2. After the parameter is enabled, you must manually disable it. Otherwise, the execution of other query statements will be affected.
3. Once this parameter is enabled, all statements of the current connection will be executed on a specified DN.
4. 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.
5. 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

5.2.8 Processing Data in a 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 requirements. 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. [Table 5-4](#) lists result set parameters. The related Connection methods are as follows:

```
// 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 5-4 Result set types

Parameter	Description
resultSetType	Type of a result set. There are three types of result sets: • 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. 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 <code>refreshRow()</code> method in a ResultSet object.
resultSetConcurrency	Concurrency type of a result set. There are two types of concurrency: • 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 the next 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 5-5](#) describes these methods.

Table 5-5 Methods for positioning a cursor 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.
first()	Places cursor to the first row.
last()	Places cursor to the last row.
absolute(int)	Places cursor to a specified row.
relative(int)	Moves the row specified by the parameter forward (that is, the value is 1, which is equivalent to next()) or backward (that is, the value is -1, which is equivalent to previous()).

Obtaining the Cursor Position from a Result Set

This cursor positioning method will 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 5-6](#) describes these methods.

Table 5-6 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.

Method	Description
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 5-7](#) describes the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 5-7 Common methods for obtaining data from a result set

Method	Description
int getInt(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as an integer.
int getInt(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as an integer.
String getString(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as a string.
String getString(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as a string.
Date getDate(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as a date.
Date getDate(String columnLabel)	Retrieves the value of the column designated by a column name in the current row as a date.

5.2.9 Closing a Database Connection

After you complete required data operations in the database, close the database connection.

To close a database connection, you can directly call the close method.

```
// 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).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

```
Connection conn = DriverManager.getConnection(sourceURL, userName, password);  
conn.close();
```

5.2.10 Log Management

The GaussDB JDBC driver uses log records to help solve problems when the GaussDB JDBC driver is used in applications. GaussDB JDBC supports the following log management methods:

1. Use the SLF4J log framework for interconnecting with applications.
2. 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.

The SLF4J may be implemented by using Log4j or Log4j2. When the Log4j is used to implement the SLF4J, the following JAR packages need to be added: **log4j-*.jar**, **slf4j-api-*.jar**, and **slf4j-log4j-*.jar** (* varies according to versions), and configuration file **log4j.properties**. 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), and configuration file **log4j2.xml**.

This method supports log management and control. The SLF4J can implement powerful log management and control functions through related configurations in files. This method is recommended.

Example:

```
public static Connection GetConnection(String username, String passwd){  
  
    String sourceURL = "jdbc:postgresql://$ip:$port/postgres?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;  
}
```

The following is an example of the **log4j.properties** file:

```
log4j.logger.org.postgresql=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
```

The following is an example of the **log4j2.xml** 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. -->
    <!-- If opengaussjdbc.jar is used, replace org.postgresql with com.huawei.opengauss.jdbc.Driver.-->
    <logger name="org.postgresql" 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.

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.

Configuration example of **logging.properties**:

```
# 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
```

The following is a code example:

```
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("org.postgresql");
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 **org.postgresql.log.Tracer** API class and specify the full name of the API implementation class in the URL.

URL example:

```
String URL = "jdbc:postgresql://$ip:$port/postgres?traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

The **org.postgresql.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 **org.postgresql.log.Tracer** API implementation class:

```
import org.postgresql.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 **traceId** 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();
    }
}
}
```

The following is an example of using **traceld**:

```
String traceld = UUID.randomUUID().toString().replaceAll("-", "");
openGaussTrace.set(traceld);
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** specified by the application cannot exceed 32 bytes. Otherwise, the extra bytes will be truncated.

5.2.11 Example: Common Operations

Example 1: Creating a Database Connection, Creating a Table, and Inserting Data

The following illustrates how to develop applications based on GaussDB JDBC APIs. Before executing the example, load the driver. For details about how to obtain and load the driver, see [JDBC Package, Driver Class, and Environment Class](#).

```
//DBTest.java
/*The following uses gsjdbc4.jar as an example.*/
// This example illustrates the main processes of JDBC-based development, covering database connection
```



```
creation, table creation, and data insertion.
// 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 = "org.postgresql.Driver";
        String sourceURL = "jdbc:postgresql://$ip:$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;
    };

    // Run a common SQL statement to create table customer_t1.
    public static void CreateTable(Connection conn) {
        Statement stmt = null;
        try {
            stmt = conn.createStatement();

            // Run a common SQL statement.
            int rc = stmt
                .executeUpdate("CREATE TABLE customer_t1 (c_customer_sk INTEGER, c_customer_name
VARCHAR(32));");

            stmt.close();
        } catch (SQLException e) {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }
}
```

```
// Run 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 customer_t1 VALUES (?,?)");
        for (int i = 0; i < 3; 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.
public static void ExecPreparedSQL(Connection conn) {
    PreparedStatement pstmt = null;
    try {
        pstmt = conn
            .prepareStatement("UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1");
        pstmt.setString(1, "new Data");
        int rowcount = pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// Run a stored procedure.
public static void ExecCallableSQL(Connection conn) {
    CallableStatement cstmt = null;
    try {
        // The stored procedure TESTPROC must be created in advance.
        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 table.
    CreateTable(conn);

    //Insert data in batches.
    BatchInsertData(conn);

    // Run a prepared statement to update data.
    ExecPreparedSQL(conn);

    // Run a stored procedure.
    ExecCallableSQL(conn);

    // Close the connection to the database.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

Example 2: High Client Memory Usage

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 damage some performance.

The cursor is valid within a transaction. Therefore, disable automatic commit and then manually commit the code.

```
// Disable automatic commit.
conn.setAutoCommit(false);
Statement st = conn.createStatement();

// Open the cursor and obtain 50 lines of data each time.
st.setFetchSize(50);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
    System.out.print("a row was returned.");
}
conn.commit();
rs.close();

// Disable the server cursor.
st.setFetchSize(0);
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
```

```
{
    System.out.print("many rows were returned.");
}
conn.commit();
rs.close();

// Close the statement.
st.close();
conn.close();
```

Run the following command to enable automatic commit:

```
conn.setAutoCommit(true);
```

Example 3: Example of Common Data Types

```
// Example of the bit type. Note that the value range of the bit type is [0,1].
Statement st = conn.createStatement();
String createsql = "create table if not exists t_bit(col_bit bit)";
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(createsql);
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 money type
// Example of using a column of the money type in the table structure.
st.execute("create table t_money(id int,col1 money)");
PreparedStatement pstmt = conn.prepareStatement("insert into t_money values(1,?)");
// Use PGObject to assign a value. The value range is [-92233720368547758.08,92233720368547758.07].
PGObject minMoney = new PGObject();
minMoney.setType("money");
minMoney.setValue("-92233720368547758.08");
pstmt.setObject(1, minMoney);
pstmt.execute();
// Use PGMoney to assign a value. The value range is [-9999999.99,9999999.99].
pstmt.setObject(1,new PGMoney(9999999.99));
pstmt.execute();
```

CAUTION

Currently, JDBC cannot call stored procedures and functions whose return data type is money.

Example 4: Obtaining the Driver Version

```
Driver.getGSVersion();
```

5.2.12 Example: Retrying SQL Queries for Applications

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.

Before executing the example, load the driver. For details about how to obtain and load the driver, see [JDBC Package, Driver Class, and Environment Class](#).

```
// The following uses gsjdbc4.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 {
    // Create a database connection.
    public static Connection GetConnection(String username, String passwd) {
        String driver = "org.postgresql.Driver";
        String sourceURL = "jdbc:postgresql://$ip:$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 an ordinary SQL statement. Create a jdbc_test1 table.
    public static void CreateTable(Connection conn) {
```

```
Statement stmt = null;
try {
    stmt = conn.createStatement();

    Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));

    // Run a common SQL statement.
    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();
}

// Run 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 = 50;
    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 process. 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 connection to the database.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
}  
  
}
```

5.2.13 Example: Importing and Exporting Data Through Local Files

When Java is used for secondary development based on GaussDB, you can use the CopyManager API to export data from the database to a local file or import a local file to the database by streaming. The file can be in CSV or text format.

The sample program is as follows. Load the GaussDB JDBC driver before executing it. For details about how to obtain and load the driver, see [JDBC Package, Driver Class, and Environment Class](#).

Prerequisites: Tables **migration_table** and **migration_table_1** have been created in the database, and some data has been inserted into the **migration_table** table.

```
// The following uses gsjdbc4.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.io.IOException;  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.sql.SQLException;  
import org.postgresql.copy.CopyManager;  
import org.postgresql.core.BaseConnection;  
  
public class Copy{  
  
    public static void main(String[] args)  
    {  
        String urls = new String("jdbc:postgresql://$ip:$port/database"); // Database URL  
        String username = System.getenv("EXAMPLE_USERNAME_ENV"); // Username  
        String password = System.getenv("EXAMPLE_PASSWORD_ENV"); // Password  
        String tablename = new String("migration_table"); // Table information  
        String tablename1 = new String("migration_table_1"); // Table information  
        String driver = "org.postgresql.Driver";  
        Connection conn = null;  
  
        try {  
            Class.forName(driver);  
            conn = DriverManager.getConnection(urls, username, password);  
        } catch (ClassNotFoundException e) {  
            e.printStackTrace(System.out);  
        } catch (SQLException e) {  
            e.printStackTrace(System.out);  
        }  
  
        // Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.  
        try {  
            copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");  
        } catch (SQLException e) {  
  
            e.printStackTrace();  
        } catch (IOException e) {
```



```
e.printStackTrace();
}
// Import data from the d:/data.txt file to the migration_table_1 table.
try {
    copyFromFile(conn, "d:/data.txt", tablename1);
} catch (SQLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

// Export the data from the migration_table_1 table to the d:/data1.txt file.
try {
    copyToFile(conn, "d:/data1.txt", tablename1);
} catch (SQLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

// Use copyIn to import data from a file to the database.
public static void copyFromFile(Connection connection, String filePath, String tableName)
    throws SQLException, IOException {
    FileInputStream fileInputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileInputStream = new FileInputStream(filePath);
        copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

// Use copyOut to export data from the database to a file.
public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
    throws SQLException, IOException {
    FileOutputStream fileOutputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileOutputStream = new FileOutputStream(filePath);
        copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
    } finally {
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.2.14 Example: Migrating Data from MySQL

The following example shows how to use CopyManager to migrate data from MySQL.

Before executing the example, load the driver. For details about how to obtain and load the driver, see [JDBC Package, Driver Class, and Environment Class](#).

```
// The following uses gsjdbc4.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.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Migration{

    public static void main(String[] args) {
        String url = new String("jdbc:postgresql://$ip:$port/database"); // Database URL
        String user = System.getenv("EXAMPLE_USERNAME_ENV");           // Database username
        String pass = System.getenv("EXAMPLE_PASSWORD_ENV");           // Database password
        String tablename = new String("migration_table"); // Table information
        String delimiter = new String("|"); // Delimiter
        String encoding = new String("UTF8"); // Character set
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer(); // Buffer to store formatted data

        try {
            // Obtain the query result set of the source database.
            ResultSet rs = getDataSet();

            // Traverse the result set and obtain records row by row.
            // The values of columns in each record are separated by the specified delimiter and end with a
            // linefeed, forming strings.
            // Add the strings to the buffer.
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                    + rs.getString(2) + delimiter
                    + rs.getString(3) + delimiter
                    + rs.getString(4)
                    + "\n");
            }
            rs.close();

            try {
                // Connect to the target database.
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                // Initialize the table.
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + "
ENCODING " + "'" + encoding + "'";
```

```
// Commit data in the buffer.
CopyManager cp = new CopyManager(baseConn);
StringReader reader = new StringReader(buffer.toString());
cp.copyIn(sql, reader);
baseConn.commit();
reader.close();
baseConn.close();
} catch (ClassNotFoundException e) {
    e.printStackTrace(System.out);
} catch (SQLException e) {
    e.printStackTrace(System.out);
}

} catch (Exception e) {
    e.printStackTrace();
}

}

//*****
// Return the query result set from the source database.
//*****
private static ResultSet getDataSet() {
    ResultSet rs = null;
    try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Connection conn = DriverManager.getConnection("jdbc:mysql://$ip:$port/database?
useSSL=false&allowPublicKeyRetrieval=true", userName, password);
        Statement stmt = conn.createStatement();
        rs = stmt.executeQuery("select * from migration_table");
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return rs;
}
}
```

5.2.15 Example: Logic Replication Code

The following example demonstrates how to use the logical replication function through the JDBC APIs. Before executing the example, load the driver. For details about how to obtain and load the driver, see [JDBC Package, Driver Class, and Environment Class](#).

For logical replication, in addition to the configuration items described in section "Logical Decoding", the following configuration items are added for streaming decoders 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 item is

set only when parallel decoding is allowed and the binary decoding is supported only in the parallel decoding scenario. For the JSON and text formats corresponding to the binary format, 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 one-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. Because there may still be a decoding statement subsequently, a 1-byte letter **P** or **F** is used as a separator between statements. **P** indicates that there are still decoded statements in this batch, and **F** indicates that this batch is completed.
5. If **C** is used in the step 3:
 1. (Optional) If the next one-byte letter is **X**, the following eight bytes (uint64) indicate the XID.
 2. (Optional) If the next one-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 one-byte letter is **P**, the batch still needs to be decoded. If the letter is **F**, the decoding of the batch ends.
6. If **I**, **U**, or **D** is used in the step 3:
 1. The next two bytes (uint16) indicate the length of the schema name.
 2. The schema name is read based on the preceding length.
 3. The next 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 one-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 next 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 next 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 a null value. 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 subsequently, 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 **standby-connection** 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 (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 number is added indicating the length of the decoding result (excluding the current uint32 number), and a uint64 number 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
Configure **max-txn-in-memory** to specify the memory threshold for caching the intermediate decoding result of a single transaction. The value ranges from 0 to 100, in MB. The default value is **0**, indicating that the memory usage is not controlled. Configure **max-reorderbuffer-in-memory** to specify the memory threshold for caching the intermediate decoding result of all transactions. The value ranges from 0 to 100, in GB. The default value is **0**, indicating that the memory usage is not controlled. When the memory usage exceeds the threshold, the intermediate decoding result is written into a temporary file during decoding, which affects the logical decoding performance.
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 GUC parameter **logical_sender_timeout**.
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. Note: 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 specified 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 user names. Multiple user names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify user names. 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** of the logical decoding connection 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.

The decoding performance (Xlog consumption) is greater than or equal to 100 Mbps in the following standard parallel decoding scenario: 16-core CPU, 128 GB memory, network bandwidth > 200 Mbps, 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.

CAUTION

The logical replication class `PGReplicationStream` is a non-thread-safe class. Concurrent calls may cause data exceptions.

```
// The following uses gsjdbc4.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).
// Logical replication function example: file name, LogicalReplicationDemo.java
// Prerequisite: Add the IP address of the JDBC user machine to the database whitelist. Add the following
// content to pg_hba.conf, and restart the database:
// Assume that the IP address of the JDBC user is 10.10.10.10.
//host all all 10.10.10.10/32 sha256
//host replication all 10.10.10.10/32 sha256

import org.postgresql.PGProperty;
import org.postgresql.jdbc.PgConnection;
import org.postgresql.replication.LogSequenceNumber;
import org.postgresql.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 = "org.postgresql.Driver";
        // Configure the IP address and haPort of the database. By default, the value of haPort is the port
        // number of the connected DN plus 1.
        String sourceURL = "jdbc:postgresql://$ip:$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: Delete a logical replication slot.
        int TEST_MODE_DROP_SLOT = 3;
        // Enable different test modes. In practice, set testMode to TEST_MODE_CREATE_SLOT to create a
        // replication slot.
        // Set testMode to TEST_MODE_START_REPL to start decoding. After decoding is complete, set
        // testMode to TEST_MODE_DROP_SLOT to delete the current replication slot.
        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")); // Specify the
            // decoding username.
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV")); // Specify the
            // corresponding password.
            // For logical replication, the following three attributes are required. Configure them as follows:
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4"); // Specify the database version.
            PGProperty.REPLICATION.set(properties, "database"); // Specify the connection used for logical
            // replication.
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple"); // Specify the protocol mode.
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
```



```
conn.getReplicationAPI()
    .createReplicationSlot()
    .logical()
    .withSlotName("replication_slot") // Name of the replication slot to be created. If the
character string contains uppercase letters, the uppercase letters are automatically converted to lowercase
letters.
    .withOutputPlugin("mppdb_decoding") // Use the mppdb_decoding plug-in.
    .make();
} else if (testMode == TEST_MODE_START_REPL) {
    // Create a replication slot before enabling this mode.
    // LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("0/2808340"); // Filter point for
decoding. Transactions committed before this LSN will not be output.
    PGReplicationStream stream = conn
        .getReplicationAPI()
        .replicationStream()
        .logical()
        .withSlotName("replication_slot") // Specify the name of the replication slot for decoding.
        .withSlotOption("include-xids", false) // Determine whether the decoding result contains
the transaction ID.
        .withSlotOption("skip-empty-xacts", true) // Determine whether to skip empty transactions.
        // .withStartPosition(waitLSN) // Set the filter point for decoding. Transactions committed
before this LSN will not be output.
        .withSlotOption("parallel-decode-num", 10) // Decoding thread concurrency
        // .withSlotOption("white-table-list", "public.t1,public.t2") // Whitelist
        // .withSlotOption("standby-connection", true) // Forcible decoding on standby
        .withSlotOption("decode-style", "t") // Decoding format
        // .withSlotOption("sending-batch", 1) // Send decoding results in batches.
        .withSlotOption("max-txn-in-memory", 100) // The memory threshold for flushing a single
decoding transaction to disks is 100 MB.
        .withSlotOption("max-reorderbuffer-in-memory", 50) // The total memory threshold for
flushing decoding transactions that are being handled to disks is 50 GB.
        .withSlotOption("exclude-users", "userA") // The logical log of the transaction executed by
user A is not returned.
        .withSlotOption("include-user", true) // The BEGIN logical log of the transaction contains
the username.
        .start();
    while (true) {
        ByteBuffer byteBuffer = stream.readPending();

        if (byteBuffer == null) {
            TimeUnit.MILLISECONDS.sleep(10L);
            continue;
        }

        int offset = byteBuffer.arrayOffset();
        byte[] source = byteBuffer.array();
        int length = source.length - offset;
        System.out.println(new String(source, offset, length));

        // If the LSN needs to be flushed, call the following APIs based on the service requirements:
        // LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
        // stream.setFlushedLSN(lastRecv);
        // stream.forceUpdateStatus();

    }
} else if (testMode == TEST_MODE_DROP_SLOT) {
    conn.getReplicationAPI()
        .dropReplicationSlot("replication_slot");
}
} catch (Exception e) {
    e.printStackTrace();
    return;
}
}
```

The following is an example of the decoding result:

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
table public t1 INSERT: a[integer]:1 b[integer]:2 c[text]:'hello'
commit xid: 15504
```

```
BEGIN CSN: 2015 first_lsn: 0/2816C20
table public t1 UPDATE: old-key: a[integer]:1 new-tuple: a[integer]:1 b[integer]:5 c[text]:'hello'
commit xid: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
table public t1 DELETE: a[integer]:1
commit xid: 15506
```

5.2.16 Example: Parameters for Connecting to the Database 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 standby cluster for DR. When the customer performs a DR failover, cluster A becomes the standby cluster for DR, 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 standby cluster for DR. Assume that cluster A is {node1,node2,node3} and cluster B is {node4,node5,node6}.

The URL can be configured as follows.

```
jdbc:postgresql://node1,node2,node3,node4,node5,node6/database?priorityServers=3
```

If you want to connect to both the primary cluster and hosts in the primary cluster, set **targetServerType** to **master**. You can configure the URL as follows:

```
jdbc:postgresql://node1,node2,node3,node4,node5,node6/database?
priorityServers=3&targetServerType=master
```

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}.

1. 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:postgresql://node1,node2,node3/database?autoBalance=true
```
2. The customer develops two applications B and C and wants 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}. The connections of application C are distributed on {node9,node10,node11,node12}. The URLs can be configured as follows.

Application A: **jdbc:postgresql://node1,node2,node3,node4,node5/
database?autoBalance=priority4**

Application B: **jdbc:postgresql://node5,node6,node7,node8,node9/
database?autoBalance=priority4**

Application C: **jdbc:postgresql://node9,node10,node11,node12,node1/database?autoBalance=priority4**

3. 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:postgresql://node1,node2,node3,node4/database?autoBalance=shuffle
```

4. If the customer does not want to use the load balancing function, configure the URL as follows.

```
jdbc:postgresql://node1/database
```

Or

```
jdbc:postgresql://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:postgresql://node1,node2,node3,node4/database?autoBalance=true&refreshCNIPListTime=3
```

Log Diagnosis

To locate faults, a customer can enable the trace log function for diagnosis. The URL can be configured as follows.

```
jdbc:postgresql://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:postgresql://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:postgresql://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:postgresql://node1/database?batchMode=on
```

5.2.17 JDBC API Reference

This section describes common JDBC APIs. For more APIs, check JDK 1.6 (software package) and JDBC 4.0.

5.2.17.1 java.sql.Connection

This section describes **java.sql.Connection**, the interface for connecting to a database.

Table 5-8 Support status for java.sql.Connection

Method Name	Return Type	Support JDBC 4
abort(Executor executor)	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
commit()	void	Yes
createArrayOf(String typeName, Object[] elements)	Array	Yes
createBlob()	Blob	Yes
createClob()	Clob	Yes
createSQLXML()	SQLXML	Yes
createStatement()	Statement	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Statement	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Statement	Yes
getAutoCommit()	Boolean	Yes
getCatalog()	String	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getHoldability()	int	Yes
getMetaData()	DatabaseMetaData	Yes
getNetworkTimeout()	int	Yes
getSchema()	String	Yes
getTransactionIsolation()	int	Yes
getTypeMap()	Map<String,Class<?>>	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
isValid(int timeout)	boolean	Yes

Method Name	Return Type	Support JDBC 4
nativeSQL(String sql)	String	Yes
prepareCall(String sql)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	CallableStatement	Yes
prepareStatement(String sql)	PreparedStatement	Yes
prepareStatement(String sql, int autoGeneratedKeys)	PreparedStatement	Yes
prepareStatement(String sql, int[] columnIndexes)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	PreparedStatement	Yes
prepareStatement(String sql, String[] columnNames)	PreparedStatement	Yes
releaseSavepoint(Savepoint savepoint)	void	Yes
rollback()	void	Yes
rollback(Savepoint savepoint)	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes
setHoldability(int holdability)	void	Yes
setNetworkTimeout(Executor executor, int milliseconds)	void	Yes

Method Name	Return Type	Support JDBC 4
setReadOnly(boolean readOnly)	void	Yes
setSavepoint()	Savepoint	Yes
setSavepoint(String name)	Savepoint	Yes
setSchema(String schema)	void	Yes
setTransactionIsolation(int level)	void	Yes
setTypeMap(Map<String,Class<?>> map)	void	Yes

NOTICE

The AutoCommit mode is used by default within the interface. 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.

5.2.17.2 java.sql.CallableStatement

This section describes **java.sql.CallableStatement**, the interface for executing the stored procedure.

Table 5-9 Support status for java.sql.CallableStatement

Method Name	Return Type	Support JDBC 4
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

Method Name	Return Type	Support JDBC 4
getDouble(int parameterIndex)	double	Yes
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.

5.2.17.3 java.sql.DatabaseMetaData

This section describes java.sql.DatabaseMetaData, the API for defining database objects.

Table 5-10 Support status for java.sql.DatabaseMetaData

Method Name	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 Name	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 Name	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 Name	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 Name	Return Type	Support JDBC 4
ownUpdatesAreVisible(int type)	boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
storesMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	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
supportsMixedCaseIdentifiers()	Boolean	Yes
supportsMultipleOpenResults()	boolean	Yes
supportsNamedParameters()	boolean	Yes
supportsOpenCursorsAcrossCommit()	boolean	Yes
supportsOpenCursorsAcrossRollback()	boolean	Yes
supportsOpenStatementsAcrossCommit()	boolean	Yes
supportsOpenStatementsAcrossRollback()	boolean	Yes

Method Name	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 Name	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 Name	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 Name	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
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

 NOTE

The `getPartitionTablePrimaryKeys(String catalog, String schema, String table)` API is used to obtain the primary key column of a partitioned table that contains global indexes. An example is as follows:

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();  
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

5.2.17.4 java.sql.Driver

This section describes **java.sql.Driver**, the database driver interface.

Table 5-11 Support status for java.sql.Driver

Method Name	Return Type	Support JDBC 4
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes

Method Name	Return Type	Support JDBC 4
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.2.17.5 java.sql.PreparedStatement

java.sql.PreparedStatement is a prepared statement API.

Table 5-12 Support status for java.sql.PreparedStatement

Method Name	Return Type	Support JDBC 4
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	Support JDBC 4
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
setObject(int parameterIndex, Object x)	void	Yes

Method Name	Return Type	Support JDBC 4
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
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes

Method Name	Return Type	Support JDBC 4
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 running **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.

5.2.17.6 java.sql.ResultSet

This section describes java.sql.ResultSet, the API for execution result sets.

Table 5-13 Support status for java.sql.ResultSet

Method Name	Return Type	Support JDBC 4
absolute(int row)	Boolean	Yes
afterLast()	void	Yes

Method Name	Return Type	Support JDBC 4
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
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes

Method Name	Return Type	Support JDBC 4
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
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes

Method Name	Return Type	Support JDBC 4
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
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes

Method Name	Return Type	Support JDBC 4
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
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes

Method Name	Return Type	Support JDBC 4
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
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes

Method Name	Return Type	Support JDBC 4
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
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes

Method Name	Return Type	Support JDBC 4
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.

5.2.17.7 java.sql.ResultSetMetaData

java.sql.ResultSetMetaData describes the information about ResultSet objects.

Table 5-14 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	Support JDBC 4
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
isSigned(int column)	boolean	Yes
isWritable(int column)	boolean	Yes

5.2.17.8 java.sql.Statement

This section describes **java.sql.Statement**, the interface for executing SQL statements.

Table 5-15 Support status for java.sql.Statement

Method Name	Return Type	Support JDBC 4
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
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

Method Name	Return Type	Support JDBC 4
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
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

Method Name	Return Type	Support JDBC 4
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.
- **LargeUpdate** methods can only be used in JDBC 4.2 or later.

5.2.17.9 javax.sql.ConnectionPoolDataSource

`javax.sql.ConnectionPoolDataSource` is the data source connection pool API.

Table 5-16 Support status for `javax.sql.ConnectionPoolDataSource`

Method Name	Return Type	Support JDBC 4
<code>getPooledConnection()</code>	<code>PooledConnection</code>	Yes
<code>getPooledConnection(String user,String password)</code>	<code>PooledConnection</code>	Yes

5.2.17.10 javax.sql.DataSource

This section describes **`javax.sql.DataSource`**, the interface for data sources.

Table 5-17 Support status for `javax.sql.DataSource`

Method Name	Return Type	Support JDBC 4
<code>getConnection()</code>	<code>Connection</code>	Yes

Method Name	Return Type	Support JDBC 4
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

5.2.17.11 javax.sql.PooledConnection

javax.sql.PooledConnection is a connection API created by a connection pool.

Table 5-18 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

5.2.17.12 javax.naming.Context

javax.naming.Context is the context API for connection configuration.

Table 5-19 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

Method Name	Return Type	Support JDBC 4
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

5.2.17.13 javax.naming.spi.InitialContextFactory

javax.naming.spi.InitialContextFactory is the initial connection context factory API.

Table 5-20 Support status for javax.naming.spi.InitialContextFactory

Method Name	Return Type	Support JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

5.2.17.14 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 **org.postgresql.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Constructor

```
public CopyManager(BaseConnection connection)
throws SQLException
```

Common Methods

Table 5-21 Common methods of CopyManager

Return Value	Method	Description	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses COPY FROM STDIN to quickly load data to tables in the database from InputStream.	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	Uses COPY FROM STDIN to quickly load data of a specified length to tables in the database from InputStream.	SQLException,IOException
long	copyIn(String sql, Reader from)	Uses COPY FROM STDIN to quickly load data to tables in the database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses COPY FROM STDIN to quickly load data of a specified length to tables in the database from Reader.	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	Sends the result set of COPY TO STDOUT from the database to the OutputStream class.	SQLException,IOException
long	copyOut(String sql, Writer to)	Sends the result set of COPY TO STDOUT from the database to the Writer class.	SQLException,IOException

5.2.17.15 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 **org.postgresql.replication** package. The declaration of the class is as follows:

```
public class PGReplicationConnection implements PGReplicationConnection
```

Constructor Method

```
public PGReplicationConnection(BaseConnection connection)
```

Common Methods

Table 5-22 Common methods of PGReplicationConnection

Return Value	Method	Description	throws
ChainedCreateReplicationSlotBuilder	createReplicationSlot()	Creates a logical replication slot.	-
void	dropReplicationSlot(String slotName)	Deletes a logical replication slot.	SQLException,IOException
ChainedStreamBuilder	replicationStream()	Logical replication is enabled.	-

5.2.17.16 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 **org.postgresql.core.v3.replication** package. The declaration of the class is as follows:

```
public class V3PGReplicationStream implements PGReplicationStream
```

Constructor

```
public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber startLSN, long updateIntervalMs, ReplicationType replicationType)
```

Common Methods

Table 5-23 Common methods of PGReplicationConnection

Return Value	Method	Description	throws
void	close()	Ends the logical replication and releases resources.	SQLException
void	forceUpdateStatus()	Forcibly sends the LSN status received, refreshed, and applied last time to the backend.	SQLException
LogSequenceNumber	getLastAppliedLSN()	Obtains the LSN when the primary node replays logs last time.	-
LogSequenceNumber	getLastFlushedLSN()	Obtains the LSN flushed by the primary node last time, that is, the LSN pushed by the current logical decoding.	-
LogSequenceNumber	getLastReceiveLSN()	Obtains the LSN received last time.	-
boolean	isClosed()	Determines whether the replication stream is disabled.	-
ByteBuffer	read()	Reads the next WAL record from the backend. If the data cannot be read, this method blocks the I/O read.	SQLException
ByteBuffer	readPending()	Reads the next WAL record from the backend. If the data cannot be read, this method does not block the I/O read.	SQLException
void	setAppliedLSN(LogSequenceNumber applied)	Sets the applied LSN.	-

Return Value	Method	Description	throws
void	setFlushedLSN(LogSequenceNumber flushed)	Sets the flushed LSN, which is sent to the backend at the next update to push the LSN on the server.	-

5.2.17.17 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 **org.postgresql.replication.fluent** package. The declaration of the class is as follows:

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

Constructor

```
public ReplicationStreamBuilder(final BaseConnection connection)
```

Common Methods

Table 5-24 Common methods of ReplicationStreamBuilder

Return Value	Method	Description	throws
ChainedLogicalStreamBuilder	logical()	Creates a logical replication stream.	-
ChainedPhysicalStreamBuilder	physical()	Creates a physical replication stream.	-

5.2.17.18 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 **org.postgresql.replication.fluent.logical** package. The declaration of this class is as follows:

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

Constructor

```
public LogicalCreateSlotBuilder(BaseConnection connection)
```

Common Methods

Table 5-25 Common methods of LogicalCreateSlotBuilder

Return Value	Method	Description	throws
T	withSlotName(String slotName)	Specifies the name of a replication slot.	-
ChainedLogicalCreateSlotBuilder	withOutputPlugin(String outputPlugin)	Plug-in name. Currently, mppdb_decoding is supported.	-
void	make()	Creates a slot with the specified parameters in the database.	SQLException
ChainedLogicalCreateSlotBuilder	self()	-	-

5.2.18 Mapping for JDBC Data Types

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

Compatib ility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
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
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	bool	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

Compatibility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
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

5.2.19 Troubleshooting

5.2.19.1 Incorrect batchMode Settings

Symptom

Set the URL parameters **batchMode=on** and **reWriteBatchedInserts=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 1:

```
// 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 thrown.  
// 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, "bbbb");  
stmt.addBatch();  
  
stmt.setInt(1, 1);  
stmt.setString(2, "aaa");  
stmt.setString(3, "bbbb");  
stmt.addBatch();  
  
stmt.setInt(1, 1);  
stmt.setString(2, "aaa");  
stmt.setString(3, "bbbb");  
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**, **batchMode** is set to **off**.

5.2.19.2 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 following error information may be reported.

Scenario 1: An error is reported.

```
"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:202)
    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
```

Solution 2: Adjust the JDK configuration on the client.

Open the `$JAVA_PATH/jre/lib/security/java.security` file and modify the following configuration items:

```
securerandom.source=file:/dev/.urandom  
securerandom.strongAlgorithms=NativePRNGNonBlocking:SUN
```

NOTICE

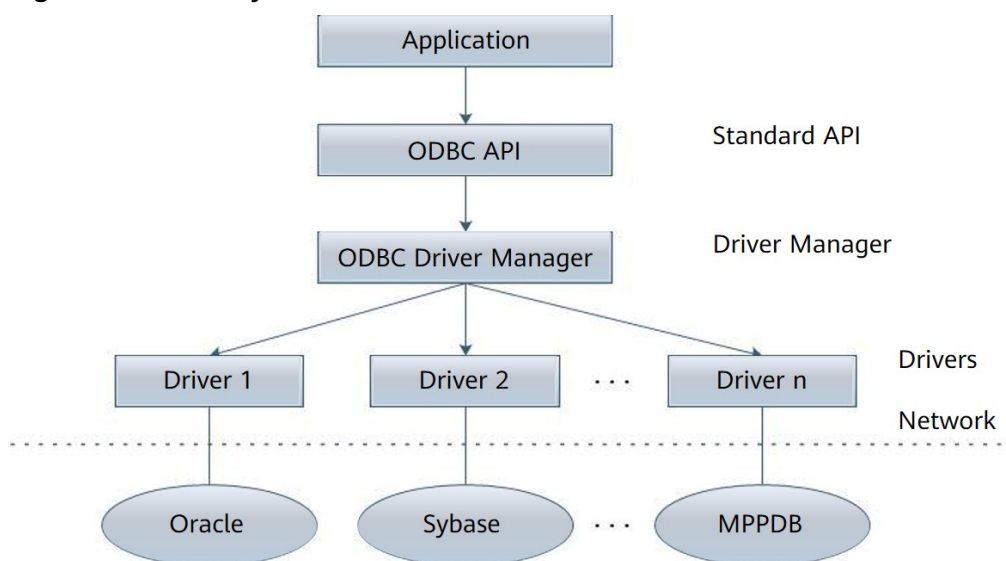
The essence of method 2 is to use pseudo-random numbers instead of strong random numbers to reduce the entropy value to be consumed. All applications that use the JDK on the client are affected. Pseudo-random numbers are used to instead of strong random numbers.

5.3 Development Based on ODBC

ODBC is a Microsoft API for accessing databases based on the X/OPEN CLI. ODBC APIs alleviate applications from directly operating in databases, and enhance the database portability, extensibility, and maintainability.

Figure 5-2 shows the system structure of ODBC.

Figure 5-2 ODBC system structure



GaussDB supports ODBC 3.5 in the following environments.

Table 5-26 OSs Supported by ODBC

OS	Platform
EulerOS 2.5	x86_64
EulerOS 2.8	Arm64
Windows 7	x86_32
Windows 7	x86_64
Windows Server 2008	x86_32

OS	Platform
Windows Server 2008	x86_64
Kylin V10	x86_64
Kylin V10	Arm64

The ODBC Driver Manager running on Unix or Linux can be unixODBC or iODBC. unixODBC-2.3.0 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 Constraints

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

5.3.1 ODBC Packages, Dependent Libraries, and Header Files

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-27](#).

Table 5-27 Driver package download list

Version	Download Address
V2.0-2.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 5-27](#).
 - 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.

ODBC Packages for the Linux OS

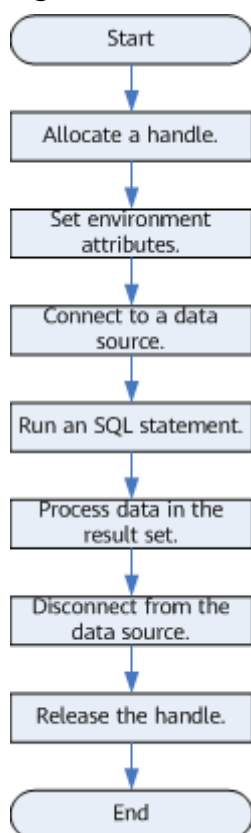
Obtain the ODBC package **GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz** from the driver package. In the Linux OS, unixODBC header files (including **sql.h** and **sqlext.h**) and the library **libodbc.so** are required in application development. These header files and library can be obtained from the unixODBC-2.3.7 installation package.

ODBC Packages for the Windows OS

Obtain the ODBC package **GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc-X86.tar.gz** from the driver package. In the Windows OS, the required header files and library files are system-resident.

5.3.2 Development Process

Figure 5-3 ODBC-based application development process



APIs Involved in the Development Process

Table 5-28 API description

Function	API
Allocate a handle	SQLAllocHandle is a generic function for allocating a handle. It can replace the following functions: • SQLAllocEnv: allocate an environment handle • SQLAllocConnect: allocate a connection handle • SQLAllocStmt: allocate a statement handle
Set environment attributes	SQLSetEnvAttr
Set connection attributes	SQLSetConnectAttr
Set statement attributes	SQLSetStmtAttr
Connect to a data source	SQLConnect

Function	API
Bind a buffer to a column in the result set	SQLBindCol
Bind the parameter marker of an SQL statement to a buffer	SQLBindParameter
Return the error message of the last operation	SQLGetDiagRec
Prepare an SQL statement for execution	SQLPrepare
Run a prepared SQL statement	SQLExecute
Run an SQL statement directly	SQLExecDirect
Fetch the next row (or rows) from the result set	SQLFetch
Return data in a column of the result set	SQLGetData
Get the column information from a result set	SQLColAttribute
Disconnect from a data source	SQLDisconnect
Release a handle	SQLFreeHandle is a generic function for releasing a handle. It can replace the following functions: • SQLFreeEnv: release an environment handle • SQLFreeConnect: release a connection handle • SQLFreeStmt: release a statement handle

 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.

 WARNING

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.

5.3.3 Configuring a Data Source in the Linux OS

The ODBC DRIVER (**psqlodbcw.so**) provided by GaussDB can be used after it has been configured in a data source. To configure a data source, you must configure the **odbc.ini** and **odbcinst.ini** files on the server. The two files are generated during the unixODBC compilation and installation, and are saved in the **/usr/local/etc** directory by default.

Procedure

Step 1 Obtain the source code package of unixODBC by clicking the following link:

Download address: <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz>

After the download, verify the integrity based on the integrity verification algorithm provided by the community.

Step 2 Install unixODBC. It does not matter if unixODBC of another version has been installed. Download the MD5 file at <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5>, view the MD5 value, and check whether the MD5 value is the same as that in the source code package.

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 3 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz** package. 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 4 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 5-29](#).

Table 5-29 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to the driver in DSN.	[DRIVER_N]
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 5-30](#).

Table 5-30 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name.	[gaussdb]
Driver	Driver name, corresponding to DriverName in odbcinst.ini .	Driver=DRIVER_N
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

Parameter	Description	Example
Password	Database user password.	Password= NOTE After a user establishes a connection, the ODBC driver automatically clears their password stored in memory. 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. 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. CAUTION The password in the configuration file must comply with the following HTTP rules: 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 handling of the percent sign (%). 2. A plus sign (+) will be replaced by a space.

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 SSL.	Sslmode=allow
Debug	If the value is 1 , the mylog file of the psqldb driver will be printed. The directory generated for storing logs is /tmp/ . If the value is 0 , no directory is generated.	Debug=1
UseServerSidePrepare	Specifies whether to enable the extended query protocol for the database. The value can be 0 or 1 . The default value is 1 , indicating that the extended query protocol is enabled.	UseServerSidePrepare=1

Parameter	Description	Example
UseBatchProtocol	Specifies whether to enable the batch query protocol. If it is enabled, DML performance can be improved. The value can be 0 or 1. The default value is 1. If the value is 0, the batch query protocol is disabled (mainly for communication with earlier database versions). If the value is 1 and support_batch_bind is set to on, the batch query protocol is enabled.	UseBatchProtocol=1
ForExtensionConnector	This parameter specifies whether the savepoint is sent. The default value is 1 . If the value is 0 , the savepoint is sent. If the value is 1 , the savepoint is not sent.	ForExtensionConnector=1
ConnectionExtraInfo	Specifies whether to display the driver deployment path and process owner in the GUC parameter connection_info .	ConnectionExtraInfo=1 NOTE The default value is 0. If the value is 1, the ODBC driver reports the driver deployment path and process owner 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.
BoolsAsChar	If this parameter is set to Yes , the Boolean value is mapped to the SQL_CHAR type. If this parameter is not set, the value is mapped to the SQL_BIT type. The default value is Yes .	BoolsAsChar = Yes

Parameter	Description	Example
RowVersioning	When an attempt is made to update a row of data, setting this parameter to Yes allows the application to detect whether the data has been modified by other users. The default value is No .	RowVersioning=Yes
ShowSystemTables	If the value is Yes , the driver regards the system catalog as a common SQL table by default. The default value is No .	ShowSystemTables=Yes
AutoBalance	Specifies whether ODBC controls load balancing. The default value is 0 , indicating that the function is disabled. 1 indicates that the function is enabled. That is, all values except 1 do not take effect.	AutoBalance=1
RefreshCNListTime	Specifies the interval for refreshing the CN list. The value is an integer and the default value is 10 , in seconds. This parameter can be configured when load balancing is enabled.	RefreshCNListTime=5

Parameter	Description	Example
Priority	This parameter can be configured when load balancing is enabled. The default value is 0 , indicating that the function is disabled. 1 indicates that the function is enabled. That is, all values except 1 do not take effect. 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.	Priority=1
UsingEip	This parameter can be configured when load balancing is enabled. The default value is 0 , indicating that the function is disabled. 1 indicates that the function is enabled. That is, all values except 1 do not take effect. 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.	UsingEip=1

Parameter	Description	Example
TcpUserTimeout	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. The unit is millisecond. The default value is 0 .	TcpUserTimeout=5000

The valid values of **Sslmode** are described in the following table.

Table 5-31 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 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.

sslmode	Enable SSL Encryption	Description
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. If they are different, modify the /etc/hosts file as user root and add the IP address and host name of the connected database node to the file. NOTE This mode does not support the default certificate of the product. Contact an administrator to generate a certificate.

Step 5 Enable the SSL mode. Contact an administrator for details.

Step 6 Configure the database server. Contact an administrator for details.

Step 7 Configure environment variables.

```
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 8 Run the following command to validate the addition:

```
source ~/.bashrc
```

----End

Verifying the Data Source Configuration

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.

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 the **ls** command 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 are missing, add the **lib** directory in 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 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 5-31](#).

- 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. For details, see [Step 6](#).

- isql: error while loading shared libraries:xxx
The dynamic library does not exist in the environment. You need to install the corresponding library.

5.3.4 Configuring a Data Source in the Windows OS

Configure an ODBC data source using the ODBC data source manager preinstalled in the Windows OS.

Procedure

Step 1 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc.tar.gz** and install **psqlodbc.exe** (for the 32-bit OS) as required.

Step 2 Open the driver manager.

Use the ODBC Driver Manager for the 32-bit OS to configure the data source. (Currently, only the ODBC Driver Manager for the 32-bit OS is supported. The following description assumes that the OS is installed on drive C. If the OS is installed on another drive, change the path accordingly.)

- For 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.

Step 3 Configure a data source.

On the **User DSN** tab, click **Add** and choose **PostgreSQL Unicode** for setup.

The screenshot shows the 'PostgreSQL ANSI ODBC Driver (psqlODBC) Setup' window. It has three tabs: 'Datasource', 'Global', and 'Manage DSN'. The 'Datasource' tab is active. The fields are as follows: Data Source is 'gaussdb1', Description is empty, Database is 'postgres', SSL Mode is 'disable', Server is redacted, Port is redacted, User Name is 'xxxx', Password is redacted, AutoBalance is '1', Refresh Time is '1', and Priority is empty. At the bottom, there are buttons for 'Test', 'Save', and 'Cancel'.

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 5-32 sslmode options and description

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.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.

sslmode	Whether SSL Encryption Is Enabled	Description
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 cert 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 cert authentication.

Step 5 Configure the GaussDB server. Contact an administrator for details.

Step 6 Run the following commands to restart the cluster:

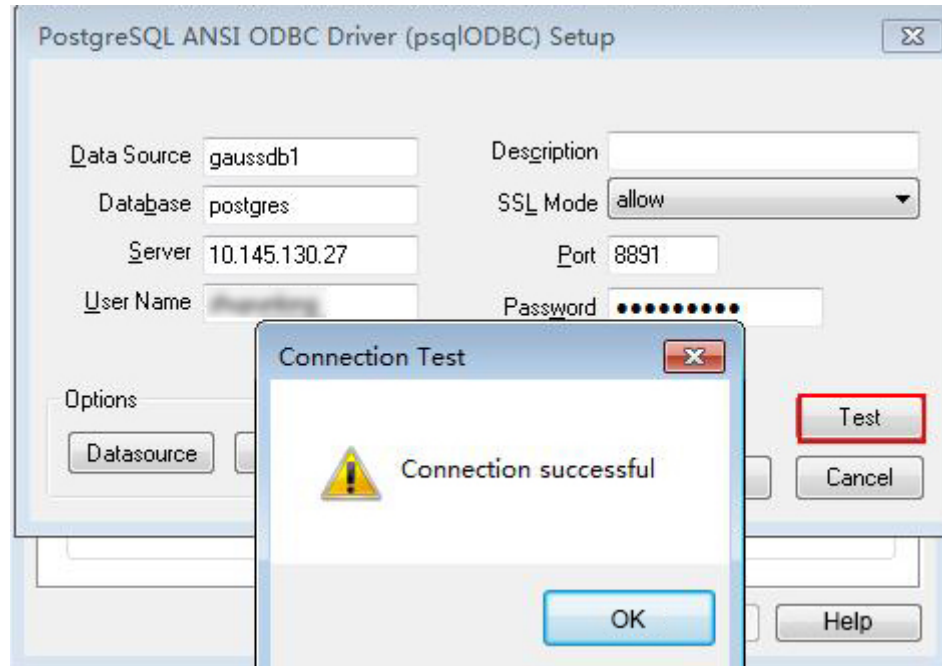
```
gs_om -t stop  
gs_om -t start
```

----End

Verifying the Data Source Configuration

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 the configuration.

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.

FAQ

- 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 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 5-32](#).

- 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. For details, see [Step 5](#).

5.3.5 Example: Common Functions and Batch Binding

Code for Common Functions

```
// The example shows how to obtain data from GaussDB through ODBC.
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
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 environment attributes (version information).
    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);
}
// 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).
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, SQL_AUTOCOMMIT_ON, 0);
// 5. Connect to the data source. 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");
// 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);
}
```

Code for Batch Processing

```
/******  
*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, data volume 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 Statement Handle  
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);  
  
    if (!SQL_SUCCEEDED(retcode)) {  
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");  
        return;  
    }  
  
    // Prepare 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 Statement  
    retcode = SQLExecute(hstmt);  
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);  
  
    if (!SQL_SUCCEEDED(retcode)) {  
        printf("SQLExecute(hstmt) failed");  
        return;  
    }  
    // Free 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;  
    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);

retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set ODBC Verion
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER*)SQL_OV_ODBC3, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetEnvAttr failed");
    goto exit;
}

// Allocate Connection
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 Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                            (SQLPOINTER)(1), 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}

// Connect to DSN
// gaussdb indicates the name of the data source used by users.
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;
}

// init table info.
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 hstmtinesrt = 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 Statement Handle
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Prepare 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, &rowsCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLRowCount failed");
        goto exit;
    }

    if (rowsCount != (batchCount - ignoreCount))
    {
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute failed");
            goto exit;
        }
    }
    else
    {
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
```

```
        printf("SQLExecute failed");
        goto exit;
    }
}

// check row number returned
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) == rowCount(%d)", process, rowCount);

    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");

// Connection
if (hdbc != SQL_NULL_HDBC) {
    SQLDisconnect(hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
```

```
// Environment
if (henv != SQL_NULL_HENV)
    SQLFreeHandle(SQL_HANDLE_ENV, henv);

return 0;
}
```

5.3.6 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
```

You only need to add the following information to the **odbc.ini** file:

```
[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.

High Performance

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.
- Set the ODBC program binding type to be the same as that in the database.
- Set the character set of the client to be the same as that in the database.
- Commit the transaction 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.)
```

Binding type case:

```
#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 committing
    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));

    //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][0], getValue[0]);
    //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][1], getValue[1]);
    i++;
}

RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
RETURN_IF_NOT(i, RowCount);
SQLCloseCursor(h_stmt);
/* Final step. Delete the table and restore the environment. */
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));

end_unit_test();
}
```

NOTE

In the 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.

Load Balancing

Load balancing can be enabled when there are a large number of concurrent applications.

- Randomly distribute concurrent connections to all CNs to enable load balancing, preventing a single CN from being overloaded and achieving high performance.

- Set **AutoBalance** to **1** to enable load balancing.
- Set **RefreshCNListTime** to **5** as required. The default refresh interval is 10 seconds.
- 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 configuration 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= (user password of the database)
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.

5.3.7 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)).

5.3.7.1 SQLAllocEnv

In ODBC 3.x, SQLAllocEnv (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle.

For details, see [SQLAllocHandle](#).

5.3.7.2 SQLAllocConnect

In ODBC 3.x, SQLAllocConnect (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle.

For details, see [SQLAllocHandle](#).

5.3.7.3 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);
```

Parameter

Table 5-33 SQLAllocHandle parameters

Keyword	Description
HandleType	Type of handle to be allocated by SQLAllocHandle. The value must be one of the following: - SQL_HANDLE_ENV (environment handle) - SQL_HANDLE_DBC (connection handle) - SQL_HANDLE_STMT (statement handle) - SQL_HANDLE_DESC (descriptor handle) The handle application sequence is: environment handle > connection handle > statement handle. The handle applied later depends on the handle applied prior to it.
InputHandle	Existing handle to use as a context for the new handle being allocated. - If HandleType is SQL_HANDLE_ENV, this parameter is set to SQL_NULL_HANDLE. - If HandleType is SQL_HANDLE_DBC, this parameter value must be an environment handle. - If HandleType is SQL_HANDLE_STMT or SQL_HANDLE_DESC, this parameter value must be a connection handle.
OutputHandlePtr	Output parameter: Pointer to a buffer that stores the returned handle in the newly allocated data structure.

Return Value

- 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 failure or connection failure.

- **SQL_INVALID_HANDLE** indicates that invalid handles were 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.

Example

See [Examples](#).

5.3.7.4 SQLAllocStmt

In ODBC 3.x, **SQLAllocStmt** (a function in ODBC 2.x) was deprecated and replaced by **SQLAllocHandle**.

For details, see [SQLAllocHandle](#).

5.3.7.5 SQLBindCol

Description

Binds application data buffers to columns in a result set.

Prototype

```
SQLRETURN SQLBindCol(SQLHSTMT StatementHandle,  
                      SQLUSMALLINT ColumnNumber,  
                      SQLSMALLINT TargetType,  
                      SQLPOINTER TargetValuePtr,  
                      SQLINTEGER BufferLength,  
                      SQLINTEGER *StrLen_or_IndPtr);
```

Parameter

Table 5-34 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.

Keyword	Description
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 TargetValuePtr buffer in bytes.
StrLen_or_IndPtr	Output parameter: pointer to the length or indicator of the buffer. If StrLen_or_IndPtr is null, no length or indicator is used.

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.6 SQLBindParameter

Description

Binds parameter markers in an SQL statement to a buffer.

Prototype

```
SQLRETURN SQLBindParameter(SQLHSTMT StatementHandle,
                            SQLUSMALLINT ParameterNumber,
                            SQLSMALLINT InputOutputType,
                            SQLSMALLINT ValueOfType,
                            SQLSMALLINT ParameterType,
                            SQLSMALLINT ColumnSize,
                            SQLSMALLINT DecimalDigits,
                            SQLPOINTER ParameterValuePtr,
                            SQLINTEGER BufferLength,
                            SQLINTEGER *StrLen_or_IndPtr);
```


Parameter

Table 5-35 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.
ValueType	C data type of the parameter.
ParameterType	SQL data type of the parameter.
ColumnSize	Size of the column or expression of the corresponding parameter marker.
DecimalDigits	Decimal digit of the column or the expression of the corresponding parameter marker.
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 Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.7 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,  
                           SQLPOINTER NumericAttributePtr);
```

Parameter

Table 5-36 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	• BufferLength indicates the length of the buffer if FieldIdentifier is an ODBC-defined field and CharacterAttributePtr points to a character 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 the FieldIdentifier field in the ColumnNumber row of the IRD is returned.

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.8 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);
```

Parameter

Table 5-37 SQLConnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from <code>SQLAllocHandle</code> .
ServerName	Name of the data source to connect.
NameLength1	Length of ServerName .
UserName	Username of the database in the data source.
NameLength2	Length of UserName .
Authentication	User password of the database in the data source.

Keyword	Description
NameLength3	Length of Authentication .

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.9 SQLDisconnect

Description

Closes the connection associated with a database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC ConnectionHandle);
```

Parameter

Table 5-38 SQLDisconnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.10 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);
```

Parameter

Table 5-39 SQLExecDirect parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
StatementText	SQL statement to be executed. Multiple statements cannot be executed at a time.
TextLength	Length of StatementText .

Return Value

- **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 failure or connection failure.
- **SQL_INVALID_HANDLE** indicates that invalid handles were 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.

Example

See [Examples](#).

5.3.7.11 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);
```

Parameter

Table 5-40 SQLExecute parameters

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.12 SQLFetch

Description

Advances the cursor to the next row of the result set and retrieves any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT StatementHandle);
```

Parameter

Table 5-41 SQLFetch parameters

Keyword	Description
StatementHandle	Statement handle, obtained from <code>SQLAllocHandle</code> .

Return Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.13 SQLFreeStmt

In ODBC 3.x, SQLFreeStmt (a function in ODBC 2.x) was deprecated and replaced with SQLFreeHandle.

For details, see [SQLFreeHandle](#).

5.3.7.14 SQLFreeConnect

In ODBC 3.x, SQLFreeConnect (a function in ODBC 2.x) was deprecated and replaced with SQLFreeHandle.

For details, see [SQLFreeHandle](#).

5.3.7.15 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);
```


Parameter

Table 5-42 SQLFreeHandle parameters

Keyword	Description
HandleType	Type of handle to be freed by SQLFreeHandle. The value must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC If HandleType is not one of the preceding values, SQLFreeHandle returns SQL_INVALID_HANDLE .
Handle	Name of the handle to be freed.

Return Value

- **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 were called. This value may also be returned by other APIs.

Precautions

If SQLFreeHandle returns **SQL_ERROR**, the handle is still valid.

Example

See [Examples](#).

5.3.7.16 SQLFreeEnv

In ODBC 3.x, SQLFreeEnv (a function in ODBC 2.x) was deprecated and replaced with SQLFreeHandle.

For details, see [SQLFreeHandle](#).

5.3.7.17 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);
```

Parameter

Table 5-43 SQLPrepare parameters

Keyword	Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of StatementText .

Return Value

- **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 failure or connection failure.
- **SQL_INVALID_HANDLE** indicates that invalid handles were 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.

Example

See [Examples](#).

5.3.7.18 SQLGetData

Description

SQLGetData is used to retrieve data for a single column in the result set. It can be called for many times to retrieve data of variable lengths.

Prototype

```
SQLRETURN SQLGetData(SQLHSTMT StatementHandle,  
                      SQLUSMALLINT Col_or_Param_Num,  
                      SQLSMALLINT TargetType,  
                      SQLPOINTER TargetValuePtr,  
                      SQLLEN BufferLength,  
                      SQLLEN *StrLen_or_IndPtr);
```

Parameter

Table 5-44 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 field in ARD. If TargetType 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 Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.19 SQLGetDiagRec

Description

Returns the current values of multiple columns 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);
```

Parameter

Table 5-45 SQLGetDiagRec parameters

Keyword	Description
HandleType	A handle-type identifier that describes the type of handle for which diagnostics are desired. The value must be one of the following: - SQL_HANDLE_ENV - SQL_HANDLE_DBC - SQL_HANDLE_STMT - SQL_HANDLE_DESC
Handle	A handle for the diagnostic data structure. Its type is indicated by HandleType . If HandleType is SQL_HANDLE_ENV , Handle may be a shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. RecNumber starts with 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.

Keyword	Description
BufferLength	Length of MessageText .
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 Value

- **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 were 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 **HandType** and **Handle** is an invalid handle.
- **SQL_ERROR** indicates that **RecNumber** is less than or equal to 0 or that **BufferLength** is smaller than 0.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call SQLGetDiagRec to obtain the **SQLSTATE** value. The possible **SQLSTATE** values are listed as follows:

Table 5-46 SQLSTATE values

SQLSTATE	Error	Description
HY000	General error.	An error occurred for which there is no specific SQLSTATE.

SQLSATATE	Error	Description
HY001	Memory allocation error.	The driver is unable to allocate memory required to support execution or completion of the function.
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.

Example

See [Examples](#).

5.3.7.20 SQLSetConnectAttr

Description

Sets connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC ConnectionHandle,  
                             SQLINTEGER Attribute,  
                             SQLPOINTER ValuePtr,  
                             SQLINTEGER StringLength);
```

Parameter

Table 5-47 SQLSetConnectAttr parameters

Keyword	Description
ConnectionHandle	Connection handle.

Keyword	Description
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 null-terminated string. If the ValuePtr parameter is a 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 Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.21 SQLSetEnvAttr

Description

Sets environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV EnvironmentHandle,
                        SQLINTEGER Attribute,
                        SQLPOINTER ValuePtr,
                        SQLINTEGER StringLength);
```

Parameter

Table 5-48 SQLSetEnvAttr parameters

Keyword	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. The value must be one of the following: • 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 Value

- **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 were 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.

Example

See [Examples](#).

5.3.7.22 SQLSetStmtAttr

Description

Sets attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT StatementHandle  
                           SQLINTEGER Attribute,  
                           SQLPOINTER ValuePtr,  
                           SQLINTEGER StringLength);
```

Parameter

Table 5-49 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 a 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 Value

- **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 were 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.

Example

See [Examples](#).

5.4 Development Based on libpq

libpq is a C API of 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. This section provides two examples to show how to write code using libpq.

5.4.1 libpq Package, Dependency Library, and Header File

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-50](#).

Table 5-50 Driver package download list

Version	Download Address
V2.0-2.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:
 - Upload the software package and verification package to the same directory on the VM.
 - 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:
 - Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
 - 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 5-50**.
 - 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 file from **GaussDB-Kernel-VxxxRxxxCxx-xxxxx-64bit-Libpq.tar.gz** in the driver package. Client programs that use libpq must include the header file **libpq-fe.h** and must connect to the libpq library.

5.4.2 Development Process

To compile and connect to a libpq source program, perform the following operations:

- Decompress the **GaussDB-Kernel-VxxxRxxxCxx-xxxxx-64bit-Libpq.tar.gz** file. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file.

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**.

- Include the **libpq-fe.h** header file.
`#include <libpq-fe.h>`
- Provide 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 the header file is located) -L (Directory where the libpq library is located) testprog.c -lpq`
- If the makefile is used, add the following option to variables **CPPFLAGS**, **LD_FLAGS**, and **LIBS**:
`CPPFLAGS += -I (Directory of the header file)
LD_FLAGS += -L (Directory of the libpq library)
LIBS += -lpq`

5.4.3 Example

Code for Common Functions

Example 1:

```
/*
 * testlibpq.c
 */
#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 was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
    * When testing connections involving cursor operations,
    * you must wrap the entire process in a transaction block instead of
    * simply using PQexec() with a basic query like "select * from pg_database".
    * This approach is too simplistic 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);
    }

    /* 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);

    /* 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);

    return 0;
}
```

Example 2:

```
/*
 * testlibpq3.c Test PQprepare
 */
#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");
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;
}
sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
paramTypes[0] = 23;
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;
    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);
}
PQfinish(conn);
return 0;
}
```

Example 3:

```
/*
 * testlibpq3.c
 * Test out-of-line parameters and binary I/Os.
 */
```

```
* 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');
*
* Expected output:
*
*
* 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 ntohs/htons */
#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 the field value may be empty). */
        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 = htonl(*(uint32_t *) iptr);

        /*
         * The binary representation of TEXT is text. Since 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;
    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)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application=test connect_timeout=5 sslmode=allow user=%s"
password=%s",
            dbname, port, hostaddr, username, password);

    /* 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);
    }

    /* 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 */

    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);
}

show_binary_results(res);

PQclear(res);

/* Close the database connection and clean up the database. */
PQfinish(conn);

return 0;
}
```

5.4.4 libpq API Reference

5.4.4.1 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. In that case, a null pointer will be returned. The interface 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.

5.4.4.1.1 PQconnectdbParams

Description

Establishes a new connection with the database server.

Prototype

```
PGconn *PQconnectdbParams(const char * const *keywords,
                           const char * const *values,
                           int expand_dbname);
```

Parameter

Table 5-51 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 expanded in this way, and any subsequent dbname value is treated as a database name.

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for 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.

Example

For details, see [Example](#).

5.4.4.1.2 PQconnectdb

Description

Establishes a new connection with the database server.

Prototype

```
PGconn *PQconnectdb(const char *conninfo);
```

Parameter

Table 5-52 PQconnectdb parameter

Keyword	Description
conninfo	Connection string. For details about the fields in the string, see Connection Parameters .

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for by the function internally.

Precautions

- This function establishes a new database connection using the parameters taken from the string **conninfo**.
- The input parameter can be empty, indicating that all default parameters can be used. It can also contain one or more parameters separated by spaces or it can contain a URL.

Example

For details, see [Example](#).

5.4.4.1.3 PQbackendPID

Supplementary Explanation

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.

5.4.4.1.4 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);
```

Parameter

Table 5-53 PQsetdbLogin parameters

Keyword	Description
pghost	Name of the host to be connected. For details, see the host field described in Connection Parameters .
pgport	Port number of the host server. For details, see the port field described in Connection Parameters .
pgoptions	Command-line options to be sent to the server during running. For details, see the options field described in Connection Parameters .
pgtty	This field can be ignored. (Previously, this field declares the output direction of server logs.)
dbName	Name of the database to be connected. For details, see the dbname field described in Connection Parameters .

Keyword	Description
login	Username for connection. For details, see the user field described in Connection Parameters .
pwd	Password used for authentication during connection. For details, see the password field described in Connection Parameters .

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for 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 = 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.

Example

For details, see [Example](#).

5.4.4.1.5 PQfinish

Description

Closes the connection to the server and releases the memory used by the PGconn object.

Prototype

```
void PQfinish(PGconn *conn);
```

Parameter

Table 5-54 PQfinish parameter

Keyword	Description
conn	Points to the 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 [Example](#).

5.4.4.1.6 PQreset

Description

Resets the communication port to the server.

Prototype

```
void PQreset(PGconn *conn);
```

Parameter

Table 5-55 PQreset parameter

Keyword	Description
conn	Points to 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.

Example

For details, see [Example](#).

5.4.4.1.7 PQstatus

Description

Returns the connection status.

Prototype

```
ConnStatusType PQstatus(const PGconn *conn);
```

Parameter

Table 5-56 PQstatus parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Value

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 attempt 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.

Example

For details, see [Example](#).

5.4.4.2 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.

5.4.4.2.1 PQexec

Description

Sends a command to the server and waits for the result.

Prototype

```
PGresult *PQexec(PGconn *conn, const char *command);
```

Parameter

Table 5-57 PQexec parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.

Return Value

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.

Example

For details, see [Example](#).

5.4.4.2 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);
```

Parameter

Table 5-58 PQprepare parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
stmtName	Prepared statement to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Value

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.

Example

For details, see [Example](#).

5.4.4.2.3 PQresultStatus

Description

Returns the result status of a command.

Prototype

```
ExecStatusType PQresultStatus(const PGresult *res);
```

Parameter

Table 5-59 PQresultStatus parameter

Keyword	Description
res	Object pointer that contains the query result.

Return Value

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

Copy Out (from the server) data transfer started.

PGRES_COPY_IN

Copy In (to the server) data transfer started.

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
Copy In/Out (to and from the server) data transfer started. 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.

Example

For details, see [Example](#).

5.4.4.2.4 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 5-60 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 submitted or even if you close the connection. To delete it, you must call PQclear. Otherwise, memory leakage occurs.

Example

For details, see [Example](#).

5.4.4.3 Functions for Asynchronous Command Processing

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 built from PQexec: PQsendQuery and PQgetResult. The functions PQsendQueryParams, PQsendPrepare, and PQsendQueryPrepared can also be used with PQgetResult.

5.4.4.3.1 PQsendQuery

Description

Sends 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 *command);
```

Parameter

Table 5-61 PQsendQuery parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
command	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.

Example

For details, see [Example](#).

5.4.4.3.2 PQsendQueryParams

Description

Sends a command and separated 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 5-62 PQsendQueryParams parameters

Keyword	Description
conn	Points to 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.

Example

For details, see [Example](#).

5.4.4.3 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 5-63 PQsendPrepare parameters

Keyword	Description
conn	Points to 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

This function is an asynchronous version of PQprepare. If it can dispatch a request, **1** is returned. Otherwise, **0** is returned. After a successful calling, call PQgetResult to check whether the server successfully created the prepared statement. The parameters of the function are processed in the same way as PQprepare. Like PQprepare, it cannot work on connections using protocol v2.0.

Example

For details, see [Example](#).

5.4.4.3.4 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 5-64 PQsendQueryPrepared parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of prepare to be executed.
nParams	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->error_message**.

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.

Example

For details, see [Example](#).

5.4.4.3.5 PQflush

Description

Tries to flush any queued output data to the server.

Prototype

```
int PQflush(PGconn *conn);
```

Parameter

Table 5-65 PQflush parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Value

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->error_message**.

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.

Example

For details, see [Example](#).

5.4.4.4 Functions for Canceling Queries in Progress

A client application can use the functions described in this section to cancel a command that is still being processed by the server.

5.4.4.4.1 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);
```

Parameter

Table 5-66 PQgetCancel parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Value

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 passed only to PQcancel or PQfreeCancel.

Example

For details, see [Example](#).

5.4.4.4.2 PQfreeCancel

Description

Releases the data structure created by PQgetCancel.

Prototype

```
void PQfreeCancel(PGcancel *cancel);
```

Parameter

Table 5-67 PQfreeCancel parameter

Keyword	Description
cancel	Points to the object pointer that contains the cancel information.

Precautions

PQfreeCancel releases a data object previously created by PQgetCancel.

Example

For details, see [Example](#).

5.4.4.4.3 PQcancel

Description

Requests the server to abandon processing of the current command.

Prototype

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

Parameters

Table 5-68 PQcancel parameters

Keyword	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.

Example

For details, see [Example](#).

5.4.5 Connection Parameters

Table 5-69 Connection parameters

Parameter	Description
host	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 .
hostaddr	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. 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: 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. NOTICE • 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.

Parameter	Description
port	Port number of the host server, or the socket file name extension for UDS connections.
user	Name of the user to be connected. By default, the username is the same as the OS name of the user running the application.
dbname	Database name. The default value is the same as the username.
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-side 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 LC_CTYPE 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	Current user identity.
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. It is useful to specify a backup value in a common tool program that wants to set a default application name but does not want it to be overwritten by the user.
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.

Parameter	Description
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.
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. 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 check_disconnect_query . If this parameter is set to on , the capability is supported. If this parameter is set to off , the capability is not supported.
sslmode	SSL encryption mode: • disable: SSL connection is not enabled. • 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 cert authentication. • verify-full: SSL connection is required. Currently, Windows ODBC does not support cert 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 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.

Parameter	Description
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). An external engine description 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.
sslcrll	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 on 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.

Parameter	Description
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 <code>.s.PGSQL.%d</code> . To use the fenced UDF mode, set this parameter to 1 . In this case, the file type is <code>.s.fencedMaster_unixdomain</code> .
replication	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: • 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. In physical or logical replication mode, only simple query protocols can be used.
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 .
connection_info	The value of connection_info is a JSON character string consisting of driver_name, driver_version, driver_path, and os_user. If the value is not null, use connection_info and ignore connectionExtraInf. 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.
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 .

5.5 Psycopg-Based Development

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 an encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both clients and servers, 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. Psycopg2 is compatible with Unicode.

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

Table 5-70 Platforms supported by Psycopg

OS	Platform	Python Version
EulerOS 2.5	Arm64 x86_64	3.8.5
EulerOS 2.9	Arm64 x86_64	3.7.4
EulerOS 2.10, Kylin v10, UnionTech20	Arm64 x86_64	3.7.9
EulerOS 2.11, SUSE 12.5	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.

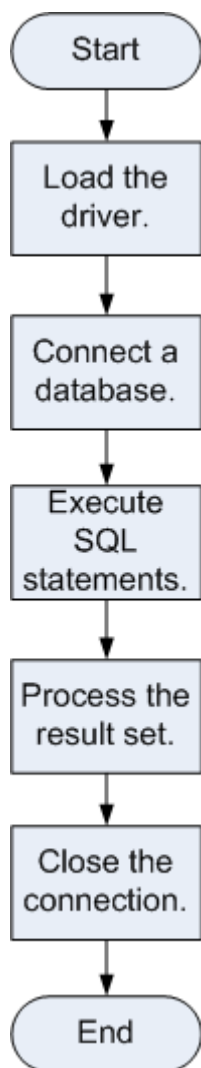
```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

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

5.5.1 Development Process

The Psycopg2-based application development process covers loading the driver, connecting to a database, executing SQL statements, processing result sets, and closing the connection.

Figure 5-4 shows the development process.

Figure 5-4 Application development process based on psycopg2

5.5.2 Psycopg Package

Obtaining the Driver Package

Download the driver package and its verification package listed in [Table 5-71](#).

Table 5-71 Driver package download list

Version	Download Address
V2.0-2.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 5-71**.
 - 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 xzvf xxx-*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 5-72](#).
 - 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 shell 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 5-72 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

5.5.3 Example: 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(database="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(database="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 pycopg2.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 = pycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
```

**CAUTION**

- 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 pycopg2 for developers to explicitly debug full SQL statements. By default, the log function is not used. This function prints SQL statements before pycopg2 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.pycopg.org/docs/extras.html?highlight=loggingconnection>.

5.5.4 Psycopg API Reference

Psycopg APIs are a set of methods provided for users. This section describes some common APIs.

5.5.4.1 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")
```

Parameter

Table 5-73 psycopg2.connect parameters

Keyword	Description
dbname	Database name.
user	Username.
password	Password.
host	Database IP address. The default type is Unix socket.
port	Connection port number. The default value is 5432 .
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 application_name .
fallback_application_name	Rollback value of application_name .
keepalives	Determines whether to enable the TCP connection on the client. The default value is 1 , indicating that the TCP connection is enabled. The value 0 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 keepalive is disabled, ignore this parameter.
keepalives_interval	Determines whether keepalive messages that are not confirmed by the server need to be resent. If keepalive is disabled, ignore this parameter.

Keyword	Description
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 1 , the data sent through the SSL connection is compressed. If this parameter is set to 0 , the compression is disabled. If no SSL connection is established, ignore this parameter.
sslcr	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.

Return Value

Connection object (for connecting to a database instance)

Example

For details, see [Example: Common Operations](#).

5.5.4.2 connection.cursor()

Description

This method returns a new cursor object.

Prototype

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

Parameter

Table 5-74 connection.cursor parameters

Keyword	Description
name	Cursor name. The default value is None .
cursor_factory	Creates a non-standard cursor. The default value is None .
scrollable	Sets the SCROLL option. The default value is None .
withhold	Sets the HOLD option. The default value is False .

Return Value

Cursor object (used for cursors that are programmed using Python in the entire database)

Example

For details, see [Example: Common Operations](#).

5.5.4.3 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 5-75 cursor.execute parameters

Keyword	Description
query	SQL statement to be executed.
vars_list	Variable list, which matches the %s placeholder in the query.

Return Value

None

Example

For details, see [Example: Common Operations](#).

5.5.4.4 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 5-76 cursor.executemany parameters

Keyword	Description
query	SQL statement that you want to execute.
vars_list	Variable list, which matches the %s placeholder in the query.

Return Value

None

Example

For details, see [Example: Common Operations](#).

5.5.4.5 connection.commit()

Function

This method commits the currently pending transaction to the database.

⚠ CAUTION

By default, Psycpg 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 [Example: Common Operations](#).

5.5.4.6 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 [Example: Common Operations](#).

5.5.4.7 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 [Example: Common Operations](#).

5.5.4.8 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 [Example: Common Operations](#).

5.5.4.9 cursor.close()

Description

This method closes the cursor of the current connection.

Prototype

```
cursor.close()
```

Parameter

None

Return Value

None

Example

For details, see [Example: Common Operations](#).

5.5.4.10 connection.close()

Function

This method closes the database connection.



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 [Example: Common Operations](#).

5.6 Commissioning

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 5-77](#) describes the adjustable configuration parameters.

Table 5-77 Configuration parameters

Parameter	Description	Value Range	Remarks
client_min_messages	Level of messages to be sent to clients.	- 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.	• 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.	• 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.	• on: A server log message is recorded when each session is connected or disconnected. • off: A server log message is not recorded when each session is connected or disconnected. Default value: off	-
log_duration	Specifies whether to record the duration of each executed statement.	• 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.	• 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.	• 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 parameter log_line_prefix .

[Table 5-78](#) describes the preceding parameter levels.

Table 5-78 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.

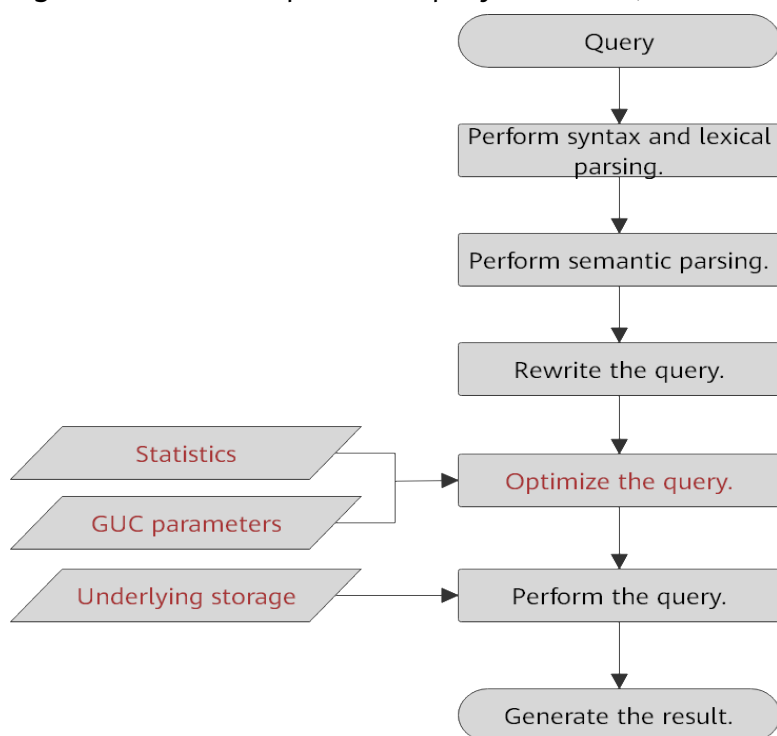
6 SQL Optimization

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 Seq Scan and a filter (that is, read every tuple and point query conditions for match). You can also use an Index Scan, which can be implemented 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.
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 are the core for optimizing a query. Accurate statistics help the planner 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 that 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 in [Figure 6-2](#).

Figure 6-2 SQL execution plan example

```
human_resource=# explain select * from hr.sections,hr.places where hr.sections.place_id = hr.places.place_id;
               QUERY PLAN
-----
Streaming (type: GATHER) (cost=6.95..22.12 rows=18 width=83) 3. Aggregation node
Node/s: All datanodes
-> Hash Join (cost=1.16..3.69 rows=3 width=83) 2. Join node
    Hash Cond: (sections.place_id = places.place_id)
    -> Streaming(type: REDISTRIBUTE) (cost=0.00..2.28 rows=3 width=25)
        Spawn on: All datanodes
        -> Seq Scan on sections (cost=0.00..1.03 rows=3 width=25) 1. Table scan node
    -> Hash (cost=1.07..1.07 rows=7 width=58)
        -> Seq Scan on places (cost=0.00..1.07 rows=7 width=58)

(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, as shown in [Figure 6-2](#).
- **pretty** indicates that the optimized display mode of GaussDB is used. A new format contains a plan node ID, directly and effectively analyzing performance, as shown in [Figure 6-3](#).
- **summary** indicates that the analysis result on this information is printed in addition to the printed information in the format specified by **pretty**.
- **run** indicates that in addition to the printed information specified by **summary**, the database exports the information as a CSV file.

Figure 6-3 Example of an execution plan using the pretty format

```
openGauss=# explain select cjxh, count(1) from dwcjk group by cjxh;
id | operation | E-rows | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----
1 | -> Row Adapter | 1 | | 52 | 58.42
2 | -> Vector Streaming (type: GATHER) | 1 | | 52 | 58.42
3 | -> Vector Hash Aggregate | 1 | 16MB | 52 | 58.02
4 | -> CStore Scan on dwcjk | 1 | 1MB | 44 | 58.00
(4 rows)
```

You can change the display format of execution plans by setting **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 details about 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 more time than a normal query. The amount of overhead depends on the nature of the query, as well as the platform being used.

Therefore, if an SQL statement is not finished after being running for a long time, run the **EXPLAIN** statement to view the execution plan and then locate the fault. If the SQL statement has been properly executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** statement to check the execution plan and information to locate the fault.

6.2.2 Description

As described in [Overview](#), **EXPLAIN** displays the execution plan, but will not actually run SQL statements. **EXPLAIN ANALYZE** and **EXPLAIN PERFORMANCE** both will actually run SQL statements and return the execution information. This section describes the execution plan and execution information in detail.

Execution Plans

The following SQL statement is used as an example:

```
select
  cjxh,
```

```
count(1)
from dwcjk
group by cjxh;
```

Run the **EXPLAIN** command and the output is as follows:

```
openGauss=# EXPLAIN SELECT * FROM t1,t2 WHERE t1.c1 = t2.c2;
               QUERY PLAN
-----
Hash Join (cost=23.73..341.30 rows=16217 width=180)
  Hash Cond: (t1.c1 = t2.c2)
    -> Seq Scan on t1 (cost=0.00..122.17 rows=5317 width=76)
    -> Hash (cost=16.10..16.10 rows=610 width=104)
        -> Seq Scan on t2 (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

Interpretation of the execution plan column (horizontal):

- **id**: ID of a node corresponding to each execution operator
- **operation**: name of an execution operator
Streaming is a special 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.
- **E-rows**: number of output rows estimated by each operator
- **E-memory**: estimated memory used by each operator on a DN. Only operators executed on DNs are displayed. In certain scenarios, the memory upper limit enclosed in parentheses will be displayed following the estimated memory usage.
- **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 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 transferring 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 level (vertical):

1. Layer 1: **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.
2. Layer 2: **Hash**

Hash operator. It is used to calculate the hash value of the operator transferred from the lower layer for subsequent hash join operations.

3. Layer 3: **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.

4. Layer 4: **Hash Join**

Join operator. It is used to join data in the **t1** and **t2** tables using the hash join method and output the result data.

Keywords in the execution plan:

1. Table access modes

– Seq Scan

Scans all rows of the table in sequence.

– Index Scan

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.

Index scans can be classified into the following types based on the index sorting mechanism:

▪ 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.
- CStore Index Ctid Scan
Scans a table based on an index and returns a qualified TID set.
- CStore Index Heap Scan
Performs the intersection, difference, and union operations of TID sets, and obtains corresponding tuple based on set results.

2. Table connection modes

- Nested Loop
A nested loop is used for queries that have a smaller data set 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 needed during the merge join, its performance excels.

3. 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.
- 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 (such as a hard-coded 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.
 - Index Cond
Specifies the index scan conditions.
 - Cstore Index And
Performs the intersection operation of TID sets with CStore Index Heap Scan.
 - Cstore Index Or
Performs the union operation of TID sets with CStore Index Heap Scan.
4. Other keywords
- Partitioned
Indicates operations on a specific partition.
 - Partition Iterator
Partition iterator, which usually indicates that a subquery is an operation on a partition.
 - InitPlan
Indicates a non-related subplan.
 - Remote Query
Indicates a query statement pushed down to a DN.
 - Exec Nodes
Indicates nodes where plans are executed.
 - Data Node Scan on

Indicates that a statement has been pushed down to a DN for execution.

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 following SQL statement is used as an example:

```
select count(1) from tb1;
```

The output of running **EXPLAIN PERFORMANCE** is as follows:

```
openGauss=# explain performance select count(1) from tb1;
id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | --> Aggregate | 19.269 | 1 | 1 | | 10KB | | | | 8 | 2.19
2 | --> Streaming (type: GATHER) | 19.191 | 4 | 4 | | 144KB | | | | 8 | 2.19
3 | --> Aggregate | [0.004,0.069] | 4 | 4 | | [10KB, 10KB] | 1MB | | | 8 | 2.03
4 | --> Seq Scan on public.tb1 | [0.001,0.060] | 5 | 5 | | [12KB, 12KB] | 1MB | | | 0 | 2.02
(4 rows)

Memory Information (identified by plan id)
-----
Coordinator Query Peak Memory:
Query Peak Memory: 0MB
DataNode Query Peak Memory
datanode1 Query Peak Memory: 0MB
datanode2 Query Peak Memory: 0MB
datanode3 Query Peak Memory: 0MB
datanode4 Query Peak Memory: 0MB
1 --Aggregate
Peak Memory: 10KB, Estimate Memory: 64MB
2 --Streaming (type: GATHER)
Peak Memory: 144KB, Estimate Memory: 64MB
3 --Aggregate
datanode1 Peak Memory: 10KB, Estimate Memory: 1024KB
datanode2 Peak Memory: 10KB, Estimate Memory: 1024KB
datanode3 Peak Memory: 10KB, Estimate Memory: 1024KB
datanode4 Peak Memory: 10KB, Estimate Memory: 1024KB
datanode1 Stream Send time: 0.000; Data Serialize time: 0.006
datanode2 Stream Send time: 0.000; Data Serialize time: 0.004
datanode3 Stream Send time: 0.000; Data Serialize time: 0.005
datanode4 Stream Send time: 0.000; Data Serialize time: 0.003
4 --Seq Scan on public.tb1
datanode1 Peak Memory: 12KB, Estimate Memory: 1024KB
datanode2 Peak Memory: 12KB, Estimate Memory: 1024KB
datanode3 Peak Memory: 12KB, Estimate Memory: 1024KB
datanode4 Peak Memory: 12KB, Estimate Memory: 1024KB
(25 rows)

Targetlist Information (identified by plan id)
-----
1 --Aggregate
Output: count((count(1)))
2 --Streaming (type: GATHER)
Output: (count(1))
Node/s: All datanodes
3 --Aggregate
Output: count(1)
4 --Seq Scan on public.tb1
Output: a, b
Distribute Key: a
(10 rows)
```

```
----- Datanode Information (identified by plan id) -----
1 --Aggregate
  (actual time=19.269..19.269 rows=1 loops=1)
  (Buffers: 0)
  (CPU: ex c/r=50593, ex row=4, ex c/c=202372, inc c/c=50094480)
2 --Streaming (type: GATHER)
  (actual time=12.371..19.191 rows=4 loops=1)
  (Buffers: 0)
  (CPU: ex c/r=12473027, ex row=4, ex c/c=49892108, inc c/c=49892108)
3 --Aggregate
  datanode1 (actual time=0.041..0.041 rows=1 loops=1)
  datanode2 (actual time=0.063..0.063 rows=1 loops=1)
  datanode3 (actual time=0.069..0.069 rows=1 loops=1)
  datanode4 (actual time=0.004..0.004 rows=1 loops=1)
  datanode1 (Buffers: shared hit=1)
  datanode2 (Buffers: shared hit=1)
  datanode3 (Buffers: shared hit=1)
  datanode4 (Buffers: 0)
  datanode1 (CPU: ex c/r=32888, ex row=1, ex c/c=32888, inc c/c=107740)
  datanode2 (CPU: ex c/r=9992, ex row=2, ex c/c=19984, inc c/c=163264)
  datanode3 (CPU: ex c/r=12272, ex row=2, ex c/c=24544, inc c/c=180008)
  datanode4 (CPU: ex c/r=0, ex row=0, ex c/c=8100, inc c/c=10768)
4 --Seq Scan on public.tbl
  datanode1 (actual time=0.027..0.029 rows=1 loops=1)
  datanode2 (actual time=0.054..0.055 rows=2 loops=1)
  datanode3 (actual time=0.059..0.060 rows=2 loops=1)
  datanode4 (actual time=0.001..0.001 rows=0 loops=1)
  datanode1 (Buffers: shared hit=1)
  datanode2 (Buffers: shared hit=1)
  datanode3 (Buffers: shared hit=1)
  datanode4 (Buffers: 0)
  datanode1 (CPU: ex c/r=74852, ex row=1, ex c/c=74852, inc c/c=74852)
  datanode2 (CPU: ex c/r=71640, ex row=2, ex c/c=143280, inc c/c=143280)
  datanode3 (CPU: ex c/r=77732, ex row=2, ex c/c=155464, inc c/c=155464)
  datanode4 (CPU: ex c/r=0, ex row=0, ex c/c=2668, inc c/c=2668)
(34 rows)

----- User Define Profiling -----
Plan Node id: 2 Track name: coordinator get datanode connection
  coordinator1: (time=0.054 total_calls=1 loops=1)
(2 rows)

===== Query Summary =====
Datanode executor start time [datanode1, datanode2]: [1.352 ms,2.276 ms]
Datanode executor end time [datanode4, datanode3]: [0.121 ms,0.187 ms]
Remote query poll time: 10.992 ms, Deserialize time: 0.000 ms
System available mem: 9188966KB
Query Max mem: 9473228KB
Query estimated mem: 2048KB
Coordinator executor start time: 0.505 ms
Coordinator executor run time: 19.277 ms
Coordinator executor end time: 0.380 ms
Planner runtime: 2.260 ms
Query Id: 79375943434081342
Total runtime: 20.803 ms
(12 rows)
```

This figure shows that the execution information can be classified into the following seven aspects.

1. The plan is displayed as a table, which contains 11 columns: **id**, **operation**, **A-time**, **A-rows**, **E-distinct**, **Peak Memory**, **E-memory**, **A-width**, **E-width**, and **E-costs**. The definition of the plan-type columns (columns started with id, operation, or E) is the same as that of running **EXPLAIN**. 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 operator. Generally, **A-time** of the operator is two values enclosed with square brackets ([]), indicating the shortest time and longest time for completing the operator on all DN, respectively.
 - **A-rows**: number of actual output tuples of the operator
 - **E-distinct**: estimated distinct value of the Hash Join operator
 - **Peak Memory**: peak memory of the operator on each DN
 - **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.
 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**, **operator 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 CNs and DNs, DN and DN connection time, and some execution information at the storage layer.
 7. **===== Query Summary =====:**
The total execution time and network traffic, including the maximum and minimum execution time in the initialization and end phases on each DN, the time in the initialization, execution, and end phases on each CN, the system available memory and statement estimation memory information during the current statement execution, are printed in this part.

NOTICE

- The difference between **A-rows** and **E-rows** shows the deviation between the optimizer estimation and actual execution. Generally, if the deviation is larger, the plan generated by the optimizer is more improper, and more manual intervention and optimization are required.
- If the difference of the **A-time** values is larger, the operator computing skew (difference between execution time on different DNs) is larger, and more manual intervention and optimization are required.
- **Max Query Peak Memory** is often used to estimate the consumed memory of SQL statements, and is also used as an important basis for setting a running memory parameter during SQL statement optimization. Generally, the output from **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** is provided for the input for further optimization.

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 a planner. 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** statement to view the execution plan and then locate the fault. If the SQL statement has been properly executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** statement 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 a planner. If no statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance.

Background

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 run 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.

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.
```

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

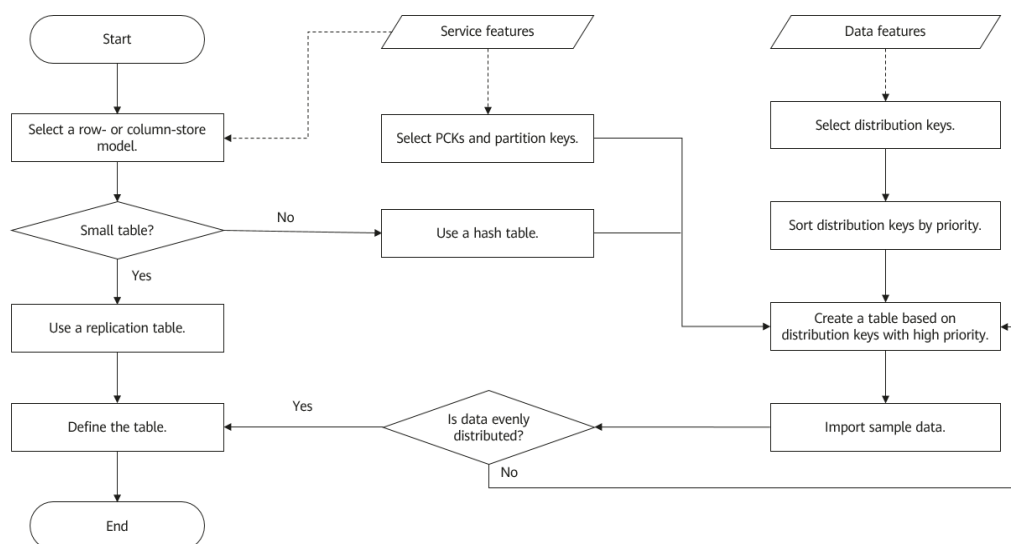
In a distributed framework, data is distributed on DN. Data on one or more DN is stored on a physical storage device. To properly define a table, you must:

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. Specifically, 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 or partial 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 6-4** 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 6-4 Procedure of defining a table



6.5.2 Selecting a Storage Model

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

Selecting a model for table storage is the first step of table definition. Select a proper storage model for your service based on the following table.

Storage Model	Application Scenario
Row store	Point queries (simple index-based queries that only return a few records) Scenarios requiring frequent addition, deletion, and modification operations

6.5.3 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.

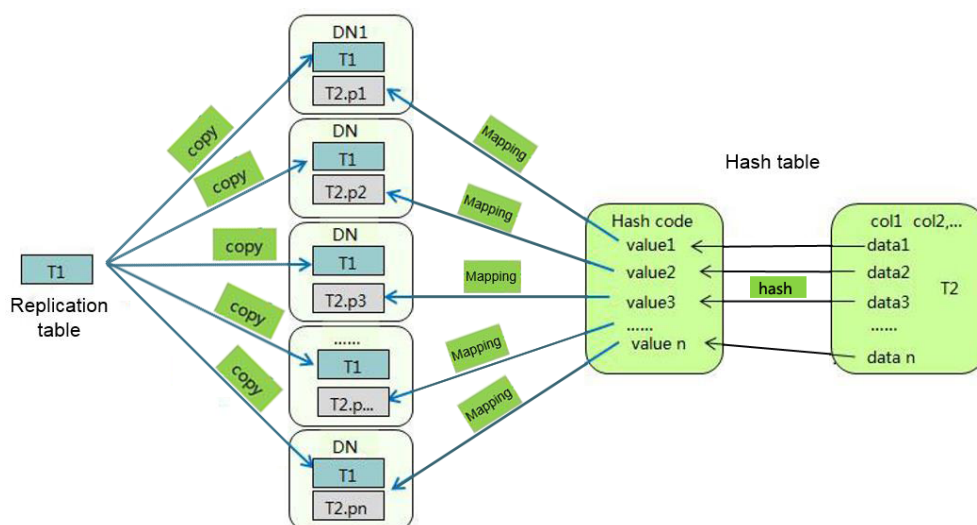
In hash mode, hash values are generated for one or more columns. You can obtain the storage location of a tuple based on the mapping between DNs and the hash values. In a hash table, I/O resources on each node can be used during data read/write, which improves the read/write speed of a table. Generally, 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 column 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 6-5](#), **T1** is a replication table and **T2** is a hash table.

Figure 6-5 Replication tables and hash tables



6.5.4 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 **dwcj** table, you are advised not 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 distributed table as required. In addition to selecting a proper distribution key, pay attention to the impact of distribution rules on data distribution.

6.5.5 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.

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.

6.5.6 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 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.

6.6.1 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 complete 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 to DN's 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" is displayed in the first line (excluding the plan format), the statement has been pushed down to DN's 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.

```
openGauss=# 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 DN's and collect data, which is inconsistent with the semantics of the **limit** clause.

```
openGauss=# 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 DN's.

```
openGauss=# 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)
```

- Delete two hash distributed tables.

```
openGauss=# DROP TABLE t;
DROP TABLE
openGauss=# DROP TABLE t1;
DROP TABLE
```

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 keys. That is, if the **join** condition matches the distribution keys 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 distributed tables.

```
openGauss=# create table t(c1 int, c2 int, c3 int) distribute by hash(c1);
CREATE TABLE
openGauss=# 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.

```
openGauss=# 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**.

```
openGauss=# 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)
```

3. Special scenarios

In some special scenarios, pushdown is not supported.

Checking Whether the Execution Plan Has Been Pushed Down to DNs

Perform the following procedure to quickly determine whether the execution plan can be pushed down to DNs:

- 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 execution plan is a distributed execution plan for sending statements and cannot be pushed down. If the execution plan contains Streaming nodes, the SQL statements can be pushed down to DNs.

For example:

```
openGauss=# explain 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:

```
openGauss=# 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);
openGauss=# CREATE TABLE test_stream(a int, b float); -- float does not support redistribution.
openGauss=# CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- The **returning** statement cannot be pushed down.

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# 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.	LOG: SQL can't be shipped, reason: RecursiveUnion contains ForeignScan is not shippable (In this table, LOG describes the cause of not supporting pushdown.) In the current version, queries containing foreign tables do not support pushdown.

No.	Scenario	Cause of Not Supporting Pushdown
2	Multiple node groups	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodengroup scenario is not shippable In the current version, pushdown is supported only when all base tables are stored and computed in the same node group.
3	ALL is not used for UNION . In this case, the return result is deduplicated.	LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches For example: <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>
4	A base table contains the system catalog.	LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable For example: <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>

No.	Scenario	Cause of Not Supporting Pushdown
5	Only VALUES is used for scanning base tables. In this case, the statement can be executed on the CN, and DNs are unnecessary.	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable For example: WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;
6	Only the recursion part has correlation conditions of correlated subqueries, and the non-recursion part has no correlation condition.	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable For example: 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;
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.	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) For example: 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;

No.	Scenario	Cause of Not Supporting Pushdown
8	recursive of multiple-layers are nested. That is, a recursive is nested in the recursion part of another recursive .	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" For example: 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;

Drop the tables.

```
openGauss=# DROP TABLE CUSTOMER1;  
DROP TABLE  
openGauss=# DROP TABLE test_stream;  
DROP TABLE  
openGauss=# 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 statement:

```
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.

Example 2: Pushing Down the Sorting Operation

For details, see [Case: Pushing Down Sort Operations to DNs](#).

6.6.2 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 RangeTblEntry and a sublink is an expression in the query parsing 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. To disable the optimization of **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.

For example:

```
openGauss=# explain 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.

For example:

```
openGauss=# explain 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 subquery is converted into such a plan.

For example:

```
gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE t2.c1 =
t1.c1);
```

QUERY PLAN

```
-----  
Seq Scan on t1  
  Filter: (SubPlan 1)  
    SubPlan 1  
      -> Seq Scan on t2  
        Filter: (c1 = t1.c1)  
(5 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 SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE  
t2.c1 = 1);
```

QUERY PLAN

```
-----  
Hash Join  
  Hash Cond: (t1.c1 = t2.c2)  
    -> Seq Scan on t1  
    -> Hash  
      -> HashAggregate  
        Group By Key: t2.c2  
        -> Seq Scan on t2  
          Filter: (c1 = 1)  
(8 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.

For example, run the following query statement:

```
gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE exists (SELECT c2 FROM  
t2 WHERE t2.c1 = t1.c1);
```

QUERY PLAN

```
-----  
Hash Semi Join  
  Hash Cond: (t1.c1 = t2.c1)  
    -> Seq Scan on t1  
    -> Hash  
      -> Seq Scan on t2  
(5 rows)
```

Replace the preceding execution plan with the following execution plan:

QUERY PLAN

```
-----  
Hash Join
```

```
Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1
-> Hash
    -> HashAggregate
        Group By Key: t2.c1
        -> 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. Equality 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. Equality 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 subquery 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. For 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 the OR clause of the EXIST related subquery connected by the OR condition is as follows:

- 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)**.
- 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 with 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);
```
- 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 correlated 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.

For 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.

For example:

```
openGauss=# 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:

```
openGauss=# 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.

For 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.

For 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, derived_table
where t1.rowid = derived_table.rowid AND
t1.c1 = derived_table.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 share scan.

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.

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.

```
openGauss=# 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:

```
openGauss=# 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.

6.6.3 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: Setting cost_param to Optimize Query Performance

See [Case: Setting cost_param and Optimizing Query Performance](#).

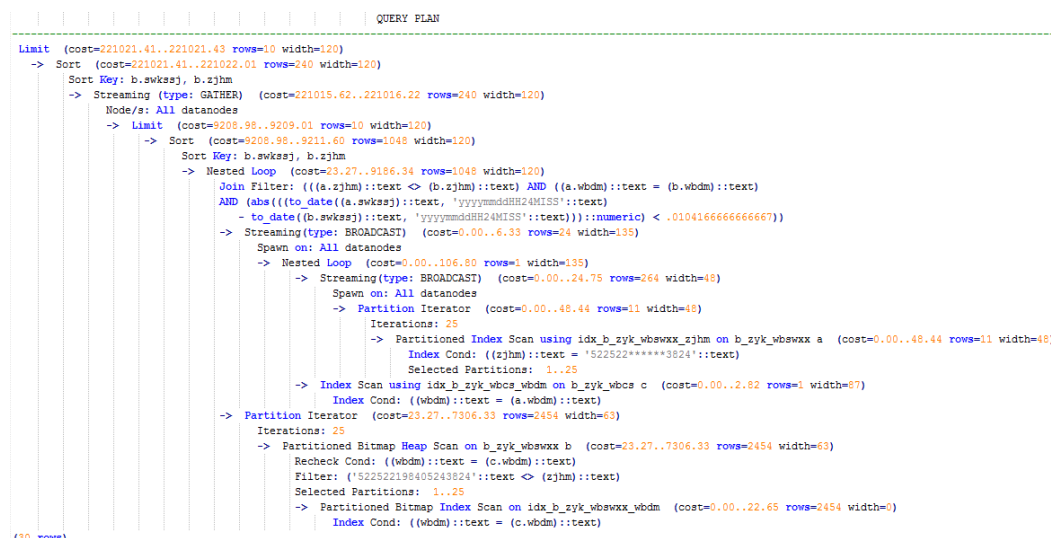
Example 3: Optimization is Not Accurate When Intermediate Results Exist in the Query Where JOIN Is Used for Multiple Tables

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 6-6 shows the execution plan. This query takes about 12s.

Figure 6-6 Using an unlogged table (1)



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)
)
;

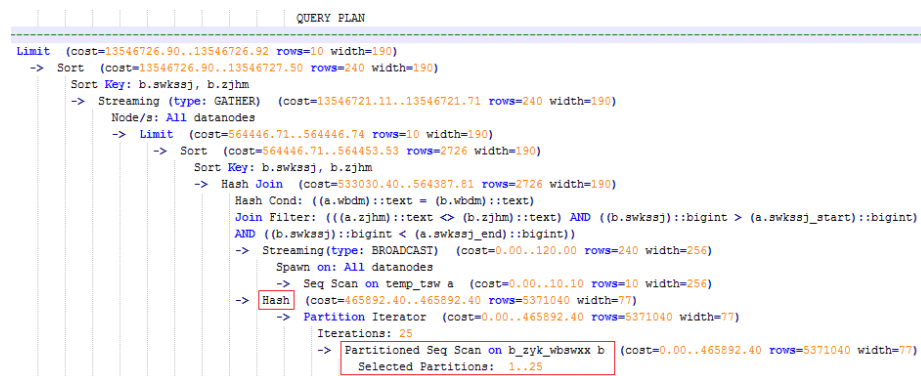
// Insert the Internet access record of the specified person, and process the start time and end time.
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
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 6-7](#) shows the execution plan.

Figure 6-7 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 equi-join 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 6-8](#) shows the execution plan. The query takes about 3s.

Figure 6-8 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..10000097345.08 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
(18 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](#).)

6.6.4 Optimizing Operators

Background

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% $[(51016-13535)/56476 \approx 66\%]$ 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)	55664.220	10000000	237060	243KB			20	2093222.75
3	-> Vector Hash Aggregate	[55124.485, 55132.180]	10000000	237060	[23548KB, 29443KB]	16MB	[20, 20]	20	[20918406.50, 20918406.50]
4	-> Vector Streaming (type: REDISTRIBUTE)	[52519.781, 53709.779]	339364604	4856184	[1219KB, 1219KB]	1MB		20	[10461210.85, 10461210.85]
5	-> Vector Hash Aggregate	[35675.636, 51016.424]	339364604	4856184	[732850KB, 746894KB]	16MB	[20, 20]	20	[10457195.65, 10457195.65]
6	-> Vector Partition Treebase	[9035.202, 13545.804]	97000000	935838097	[9KB, 9KB]	1MB		20	[10195891.68, 10195891.68]
7	-> Partitioned CStore Scan on xuj1.e_gp_day_energy_mv_1	[9015.645, 13535.346]	97000000	935838097	[845KB, 845KB]	1MB		20	[10195891.68, 10195891.68]

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.

```
openGauss=# explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
```

id	operation	A-time	A-rows	Peak Memory	A-width
1	-> Streaming (type: GATHER)	3666.020	3360	195KB	
2	-> Seq Scan on store_sales	[3594.611, 3594.611]	3360	[34KB, 34KB]	

(2 rows)

Predicate Information (identified by plan id)

```
2 --Seq Scan on store_sales  
  Filter: (ss_sold_date_sk = 2450944)  
  Rows Removed by Filter: 4968936
```

```
openGauss=# create index idx on store_sales_row(ss_sold_date_sk);  
CREATE INDEX
```

```
openGauss=# explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk = 2450944;
```

id	operation	A-time	A-rows	Peak Memory	A-width
----	-----------	--------	--------	-------------	---------

```
+-----+-----+-----+-----+
1 | -> Streaming (type: GATHER) | 81.524 | 3360 | 195KB |
2 | -> Index Scan using idx on store_sales_row | [13.352,13.352] | 3360 | [34KB, 34KB] |
(2 rows)
```

In this example, the full table scan filters much data and returns 3360 records. After an index has been created on the **ss_sold_date_sk** column, the scanning efficiency is significantly boosted from 3.6s to 13 ms by using **IndexScan**.

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 181s. 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 takes more than 200 ms.

```
openGauss=# explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+
1 | -> Row Adapter | 184300.301 | 1 | 1 | 11KB | | | | 0 | 48629179.77
2 | -> Vector Aggregate | 184300.280 | 1 | 1 | 181KB | | | | 0 | 48629179.77
3 | -> Vector Streaming (type: GATHER) | 184300.186 | 4 | 4 | 189KB | | | | 0 | 48629179.77
4 | -> Vector Aggregate | [165575.384,184252.368] | 4 | 4 | [140KB, 140KB] | 1MB | | | 0 | 48629179.61
5 | -> Vector Nest Loop (6,7) | [162918.848,181438.162] | 2880404 | 2880404 | [74KB, 74KB] | 1MB | | | 0 | 48627379.35
6 | -> CStore Scan on store_sales ss | [15.660,16.229] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | | | 4 | 16683.10
7 | -> Vector Materialize | [118314.521,132478.454] | 12968211302 | 18000 | [869KB, 900KB] | 16MB | [8,8] | 4 | 3880.00
8 | -> CStore Scan on item i | [0.234,0.243] | 18000 | 18000 | [476KB, 476KB] | 1MB | | | 4 | 3867.50
(8 rows)
```

```
openGauss=# set enable_nestloop=off;
SET
```

```
openGauss=# set enable_mergejoin=off;
SET
```

```
openGauss=# explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+
1 | -> Row Adapter | 291.066 | 1 | 1 | 11KB | | | | 0 | 32308.66
2 | -> Vector Aggregate | 291.052 | 1 | 1 | 181KB | | | | 0 | 32308.66
3 | -> Vector Streaming (type: GATHER) | 290.973 | 4 | 4 | 189KB | | | | 0 | 32308.66
4 | -> Vector Aggregate | [220.792,234.532] | 4 | 4 | [140KB, 140KB] | 1MB | | | 0 | 32308.50
5 | -> Vector Hash Join (6,7) | [209.987,223.345] | 2880404 | 2880404 | [236KB, 241KB] | 16MB | [8,8] | 0 | 30508.24
6 | -> CStore Scan on store_sales ss | [13.132,13.717] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | | | 4 | 16683.10
7 | -> CStore Scan on item i | [0.214,0.246] | 18000 | 18000 | [477KB, 477KB] | 1MB | | | 4 | 3867.50
(7 rows)
```

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**.

```
openGauss=# explain analyze select count(*) from store_sales group by ss_item_sk;
id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+
1 | -> Row Adapter | 1977.385 | 18000 | 17644 | 20KB | | | | 4 | 92875.24
2 | -> Vector Streaming (type: GATHER) | 1973.617 | 18000 | 17644 | 1946KB | | | | 4 | 92875.24
3 | -> Vector Sort Aggregate | [1784.800,1893.243] | 18000 | 17644 | [273KB, 273KB] | 1MB | | | 4 | 92106.02
4 | -> Vector Sort | [1752.270,1848.357] | 2880404 | 2880404 | [128466KB, 135135KB] | 16MB | [8,8] | 4 | 88541.40
5 | -> CStore Scan on store_sales | [12.483,13.548] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | | | 4 | 16683.10
(5 rows)
```

```
openGauss=# set enable_sort=off;
SET
```

```
openGauss=# explain analyze select count(*) from store_sales group by ss_item_sk;
id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+
1 | -> Row Adapter | 838.215 | 18000 | 17644 | 20KB | | | | 4 | 21016.93
2 | -> Vector Streaming (type: GATHER) | 834.264 | 18000 | 17644 | 229KB | | | | 4 | 21016.93
3 | -> Vector Hash Aggregate | [585.017,758.204] | 18000 | 17644 | [262552KB, 262564KB] | 16MB | [8,8] | 4 | 20327.72
4 | -> CStore Scan on store_sales | [12.540,13.941] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | | | 4 | 16683.10
(4 rows)
```

6.6.5 Optimizing Data Skew

Data skew breaks the balance among nodes in the distributed MPP architecture. If the amount of data stored or processed by a node is much greater than that by other nodes, the following problems may occur:

- Storage skew severely limits the system capacity. The skew on a single node hinders system storage utilization.
- Computing skew severely affects performance. The data to be processed on the skew node is much more than that on other nodes, deteriorating overall system performance.
- Data skew severely affects the scalability of the MPP architecture. During storage or computing, data with the same values is often placed on the same

node. Therefore, even if we add nodes after a data skew occurs, the skew data (data with the same values) is still placed on the node and affects the system capacity or performance bottleneck.

GaussDB provides a complete solution for data skew, including storage and computing skew.

Data Skew in the Storage Layer

In the GaussDB database, data is distributed and stored on each DN. You can improve the query efficiency by using distributed execution. However, if data skew occurs, bottlenecks exist on some DNs during distribution execution, affecting the query performance. This is because the distribution key is not properly selected. This can be solved by adjusting the distribution key.

For example:

```
openGauss=# explain performance select count(*) from inventory;
5 --CStore Scan on lmz.inventory
   dn_6001_6002 (actual time=0.444..83.127 rows=42000000 loops=1)
   dn_6003_6004 (actual time=0.512..63.554 rows=27000000 loops=1)
   dn_6005_6006 (actual time=0.722..99.033 rows=45000000 loops=1)
   dn_6007_6008 (actual time=0.529..100.379 rows=51000000 loops=1)
   dn_6009_6010 (actual time=0.382..71.341 rows=36000000 loops=1)
   dn_6011_6012 (actual time=0.547..100.274 rows=51000000 loops=1)
   dn_6013_6014 (actual time=0.596..118.289 rows=60000000 loops=1)
   dn_6015_6016 (actual time=1.057..132.346 rows=63000000 loops=1)
   dn_6017_6018 (actual time=0.940..110.310 rows=54000000 loops=1)
   dn_6019_6020 (actual time=0.231..41.198 rows=21000000 loops=1)
   dn_6021_6022 (actual time=0.927..114.538 rows=54000000 loops=1)
   dn_6023_6024 (actual time=0.637..118.385 rows=60000000 loops=1)
   dn_6025_6026 (actual time=0.288..32.240 rows=15000000 loops=1)
   dn_6027_6028 (actual time=0.566..118.096 rows=60000000 loops=1)
   dn_6029_6030 (actual time=0.423..82.913 rows=42000000 loops=1)
   dn_6031_6032 (actual time=0.395..78.103 rows=39000000 loops=1)
   dn_6033_6034 (actual time=0.376..51.052 rows=24000000 loops=1)
   dn_6035_6036 (actual time=0.569..79.463 rows=39000000 loops=1)
```

In the performance information, the number of scan lines on each DN in the inventory table is displayed. The maximum number of scan lines is 63000000, and the minimum number is 15000000, which is four times of the actual number. This value difference on the performance of data scan is acceptable, but if the join operator exists in the upper-layer, the impact on the performance cannot be ignored.

Generally, the data table is hash distributed on each DN; therefore, it is important to choose a proper distribution key. Run **table_skewness()** to view data skew of each DN in the inventory table. The query result is as follows:

```
openGauss=# select table_skewness('inventory');
table_skewness
-----
("dn_6015_6016",63000000,8.046%)
("dn_6013_6014",60000000,7.663%)
("dn_6023_6024",60000000,7.663%)
("dn_6027_6028",60000000,7.663%)
("dn_6017_6018",54000000,6.897%)
("dn_6021_6022",54000000,6.897%)
("dn_6007_6008",51000000,6.513%)
("dn_6011_6012",51000000,6.513%)
("dn_6005_6006",45000000,5.747%)
("dn_6001_6002",42000000,5.364%)
("dn_6029_6030",42000000,5.364%)
("dn_6031_6032",39000000,4.981%)
```

```
("dn_6035_6036",39000000,4.981%)
("dn_6009_6010",36000000,4.598%)
("dn_6003_6004",27000000,3.448%)
("dn_6033_6034",24000000,3.065%)
("dn_6019_6020",21000000,2.682%)
("dn_6025_6026",15000000,1.916%)
(18 rows)
```

The table definition indicates that the table uses the **inv_date_sk** column as the distribution key, which causes a data skew. Based on the data distribution of each column, change the distribution key to **inv_item_sk** when creating a table. The skew status is as follows:

```
openGauss=# select table_skewness('inventory');
      table_skewness
-----
("dn_6001_6002",43934200,5.611%)
("dn_6007_6008",43829420,5.598%)
("dn_6003_6004",43781960,5.592%)
("dn_6031_6032",43773880,5.591%)
("dn_6033_6034",43763280,5.589%)
("dn_6011_6012",43683600,5.579%)
("dn_6013_6014",43551660,5.562%)
("dn_6027_6028",43546340,5.561%)
("dn_6009_6010",43508700,5.557%)
("dn_6023_6024",43484540,5.554%)
("dn_6019_6020",43466800,5.551%)
("dn_6021_6022",43458500,5.550%)
("dn_6017_6018",43448040,5.549%)
("dn_6015_6016",43247700,5.523%)
("dn_6005_6006",43200240,5.517%)
("dn_6029_6030",43181360,5.515%)
("dn_6025_6026",43179700,5.515%)
("dn_6035_6036",42960080,5.487%)
(18 rows)
```

Data skew is solved.

In addition to the **table_skewness()** view, you can use the **table_distribution** function and the **PGXC_GET_TABLE_SKEWNESS** view to efficiently query the data skew status of each table.

Data Skew in the Computing Layer

Even if data is balanced across nodes after you change the distribution key of a table, data skew may still occur during a query. If data skew occurs in the result set of an operator on a DN, skew will also occur during the computing that involves the operator. Generally, this is caused by data redistribution during the execution.

During a query, **JOIN** keys and **GROUP BY** keys are not used as distribution keys. Data is redistributed among DNs based on the hash values of data on the keys. The redistribution is implemented using the Redistribute operator in an execution plan. Data skew in redistribution keys can lead to data skew during system operation. After the redistribution, some nodes will have much more data, process more data, and will have much lower performance than others.

In the following example, the **s** and **t** tables are joined, and **s.x** and **t.x** columns in the join condition are not their distribution keys. Table data is redistributed using the **REDISTRIBUTE** operator. Data skew occurs in the **s.x** column and not in the **t.x** column. The result set of the **Streaming** operator (**id** being **6**) on datanode2 has data three times that of other DNs and causes a skew.

```

openGauss=# explain select * from skew s,test t where s.x = t.x order by s.a limit 1;
id |          operation          |  A-time
----+-----+-----
1 | -> Limit                    | 52622.382
2 | -> Streaming (type: GATHER) | 52622.374
3 | -> Limit                    | [30138.494,52598.994]
4 | -> Sort                     | [30138.486,52598.986]
5 | -> Hash Join (6,8)          | [30127.013,41483.275]
6 | -> Streaming(type: REDISTRIBUTE) | [11365.110,22024.845]
7 | -> Seq Scan on public.skew s | [2019.168,2175.369]
8 | -> Hash                     | [2460.108,2499.850]
9 | -> Streaming(type: REDISTRIBUTE) | [1056.214,1121.887]
10 | -> Seq Scan on public.test t | [310.848,325.569]
(10 rows)
6 --Streaming(type: REDISTRIBUTE)
   datanode1 (rows=5050368)
   datanode2 (rows=15276032)
   datanode3 (rows=5174272)
   datanode4 (rows=5219328)

```

Computing skew is more difficult to detect than storage skew. To solve computing skew, GaussDB provides the Runtime Load Balance Technology (RLBT) solution, controlled by the GUC parameter **skew_option**. The RLBT solution addresses how to detect and solve data skew.

1. Detect data skew.

The solution first checks whether skew data exists in redistribution keys used for computing. RLBT can detect data skew based on statistics, specified hints, or rules.

– Detection based on statistics

Run the **ANALYZE** statement to collect statistics on tables. The optimizer will automatically identify skew data on redistribution keys based on the statistics and generate optimization plans for queries having potential skew. When the redistribution key has multiple columns, statistics information can be used for identification only when all columns belong to the same base table.

The statistics information can only provide the skew of the base table. When a column in the base table is skewed, other columns have filter conditions, or after the join of other tables, the skewed data may still exist on the skewed column. If **skew_option** is **normal**, it indicates that the skew data still exists, and the base tables will be optimized to solve skew. If **skew_option** is **lazy**, it indicates that no more skew data exists and the optimization will stop.

– Detection based on specified hints

The intermediate results of complex queries are difficult to estimate based on statistics. In this case, you can specify hints to provide the skew information based on which the optimizer optimizes queries. For details about the syntax of hints, see [Skew Hints](#).

– Detection based on rules

In a business intelligence (BI) system, a large number of SQL statements having outer joins (including left joins, right joins, and full joins) are generated, and many NULL values will be generated in empty columns that have no match for outer joins. If JOIN or GROUP BY operations are performed on the columns, data skew will occur. RLBT can automatically identify this scenario and generate an optimization plan for NULL value skew.

2. Solve computing skew.

Join and **Aggregate** operators are optimized to solve skew.

– **Join** optimization

Skew and non-skew data is separately processed. Details are as follows:

a. When redistribution is required on both sides of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side with skew. Specifically, perform round-robin on skew data and redistribution on non-skew data.

Use **PART_REDISTRIBUTE_PART_BROADCAST** on the side with no skew. Specifically, perform broadcast on skew data and redistribution on non-skew data.

b. When redistribution is required on only one side of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side where redistribution is required.

Use **PART_LOCAL_PART_BROADCAST** on the side where redistribution is not required. Specifically, perform broadcast on skew data and retain other data locally.

c. When a table has NULL values padded:

Use **PART_REDISTERIBUTE_PART_LOCAL** on the table. Specifically, retain the NULL values locally and perform redistribution on other data.

In the example query, the **s.x** column contains skewed data and its value is **0**. The optimizer identifies the skew data in statistics and generates the following optimization plan:

id	operation	A-time
1	-> Limit	23642.049
2	-> Streaming (type: GATHER)	23642.041
3	-> Limit	[23310.768,23618.021]
4	-> Sort	[23310.761,23618.012]
5	-> Hash Join (6,8)	[20898.341,21115.272]
6	-> Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)	[7125.834,7472.111]
7	-> Seq Scan on public.skew s	[1837.079,1911.025]
8	-> Hash	[2612.484,2640.572]
9	-> Streaming(type: PART REDISTRIBUTE PART BROADCAST)	[1193.548,1297.894]
10	-> Seq Scan on public.test t	[314.343,328.707]
(10 rows)		
5 --Hash Join (6,8)	Hash Cond: s.x = t.x	
	Skew Join Optimized by Statistic	
6 --Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)	datanode1 (rows=7635968)	
	datanode2 (rows=7517184)	
	datanode3 (rows=7748608)	
	datanode4 (rows=7818240)	

In the preceding execution plan, **Skew Join Optimized by Statistic** indicates that this is an optimized plan used for handling data skew. The **Statistic** keyword indicates that the plan optimization is based on statistics; **Hint** indicates that the optimization is based on hints; **Rule** indicates that the optimization is based on rules. In this plan, skew and non-skew data are separately processed. Non-skew data in the **s** table is redistributed based on its hash values, and skew data (whose value is **0**) is evenly distributed on all nodes in round-robin mode. In this way, data skew is solved.

To ensure result correctness, the **t** table also needs to be processed. In the **t** table, the data whose value is **0** (skew value in the **s.x** table) is broadcast and other data is redistributed based on its hash values.

In this way, data skew in **JOIN** operations is solved. The above result shows that the output of the **Streaming** operator (**id** being **6**) on each DN is balanced and the end-to-end performance of the query is doubled.

– **Aggregate optimization**

For aggregation, distribution keys on each DN are deduplicated based on the **GROUP BY** key and then redistributed. After the deduplication on DNs, each DN has only one value for redistribution key. The global occurrences of each value will not be greater than the number of DNs. Therefore, no serious data skew will occur. Take the following query as an example:

```
select c1, c2, c3, c4, c5, c6, c7, c8, c9, count(*) from t group by c1, c2, c3, c4, c5, c6, c7, c8, c9 limit 10;
```

The command output is as follows:

id	operation	A-time	A-rows
1	-> Streaming (type: GATHER)	130621.783	12
2	-> GroupAggregate	[85499.711,130432.341]	12
3	-> Sort	[85499.509,103145.632]	36679237
4	-> Streaming(type: REDISTRIBUTE)	[25668.897,85499.050]	36679237
5	-> Seq Scan on public.t	[9835.069,10416.388]	36679237

(5 rows)

4 --Streaming(type: REDISTRIBUTE)
datanode1 (rows=36678837)
datanode2 (rows=100)
datanode3 (rows=100)
datanode4 (rows=200)

A large amount of skew data exists. As a result, after data is redistributed based on its **GROUP BY** key, the data volume of datanode1 is hundreds of thousands of times that of others. After optimization, a **GROUP BY** operation is performed on the DN to deduplicate data. After redistribution, no data skew occurs.

id	operation	A-time
1	-> Streaming (type: GATHER)	10961.337
2	-> HashAggregate	[10953.014,10953.705]
3	-> HashAggregate	[10952.957,10953.632]
4	-> Streaming(type: REDISTRIBUTE)	[10952.859,10953.502]
5	-> HashAggregate	[10084.280,10947.139]
6	-> Seq Scan on public.t	[4757.031,5201.168]

(6 rows)

Predicate Information (identified by plan id)

3 --HashAggregate
Skew Agg Optimized by Statistic
(2 rows)

4 --Streaming(type: REDISTRIBUTE)
datanode1 (rows=17)
datanode2 (rows=8)
datanode3 (rows=8)
datanode4 (rows=14)

Applicability

– **Join operator**

- **nest loop, merge join, and hash join** can be optimized.

- If skew data is on the left to the join, **inner join**, **left join**, **semi join**, and **anti join** are supported. If skew data is on the right to the join, **inner join**, **right join**, **right semi join**, and **right anti join** are supported.
- For an optimization plan generated based on statistics, the optimizer checks whether it is optimal by estimating its cost. Optimization plans based on hints or rules are forcibly generated.
- **Aggregate operator**
 - **array_agg**, **string_agg**, and **subplan in agg qual** cannot be optimized.
 - A plan generated based on statistics is affected by its cost, the **plan_mode_seed** parameter, and the **best_agg_plan** parameter. A plan generated based on hints or rules are not affected by them.

6.7 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, if the **t2.d2** column does not contain null (the **t2.d2** column is not null in the table definition), the query can be modified as follows:

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
```

The generated execution plan is as follows:

Figure 6-9 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 **!=** or **<>** operators, **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.

6.8 Configuring Key Parameters for SQL Optimization

This section describes key CN configuration parameters that affect GaussDB SQL performance optimization.

Table 6-2 CN parameters

Parameter/ Reference Value	Description
enable_nestloop=off	Specifies how the optimizer uses nested-loop join. If this parameter is set to on, the optimizer preferentially uses nested-loop join. If it is set to off, the optimizer preferentially uses other methods, if any. NOTE To temporarily change the value of this parameter for the current database connection (session), execute the following SQL statement: SET enable_nestloop to off; Disabling this parameter does not affect the index nested-loop join operator (corresponding to the enable_index_nestloop parameter). During real-world optimization, it is important to assess whether to disable this parameter based on the specific circumstances. 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.
enable_index_nestloop=on	Specifies whether to enable the index nested-loop join operator. NOTE To temporarily change the value of this parameter for the current database connection (session), execute the following SQL statement: SET enable_index_nestloop to off; Differences between enable_index_nestloop and enable_nestloop: enable_index_nestloop helps manage scenarios where the inner table undergoes index scans, and the outer table supplies index condition parameters for the inner table.

Parameter/ Reference Value	Description
enable_bitmapscan=on	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. NOTE To temporarily change the value of this parameter for the current database connection (session), execute the following SQL statement: SET enable_bitmapscan to off; 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 optimization, 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.
enable_fast_query_shipping=on	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. NOTE To temporarily change the value of this parameter for the current database connection (session), execute the following SQL statement: SET enable_fast_query_shipping to off;
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_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.
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.

Parameter/ Reference Value	Description
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 rewriting rules LAZYAGG, MAGICSET, PARTIALPUSH, UNIQUECHECK, DISABLEREP, INTARGETLIST, PREDPUSHNORMAL, PREDPUSHFORCE, PREDPUSH, DISABLE_PULLUP_EXPR_SUBLINK, DISABLE_PULLUP_NOT_IN_SUBLINK, and ENABLE_SUBLINK_PULLUP_ROWNUM.
sql_beta_feature	Specifies whether the optimizer enables the beta features of 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, and DISABLE_BITMAP_COST_WITH_LOSSY_PAGES.

6.9 Hint-based Tuning

6.9.1 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**, and **indexonlyscan** hints.
- Sublink name hints.
- Skew hints, supporting only the skew in the redistribution involving Join or HashAgg.
- Hints of GUC parameters, which take effect in a query and do not take effect 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.

Precautions

- Hints do not support **Agg**, **Sort**, **Setop**, or **Subplan**.
- Hints do not support SMP or Node Groups.

Example

The following is the original plan and is used for comparing with the optimized ones:

```
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))
-> Index Scan using customer_pkey on customer (cost=0.00..3.35
rows=1 width=8)
Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
-> Index Only Scan using promotion_pkey on promotion (cost=0.00..3.35
rows=1 width=4)
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)
-> Index Scan using customer_address_pkey on customer_address ad2 (cost=0.00..3.35
rows=1 width=368)
Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(34 rows)
```

6.9.2 Join Order Hints

Description

Specifies the join order and outer/inner tables.

Syntax

- Specify only the join order.

```
leading(join_table_list)
```

- Specify the join order and outer/inner tables. The outer/inner tables are specified by the outermost parentheses.

```
leading((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.

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 order 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 [Example](#) 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-memory	E-width	E-costs
1	-> Row Adapter	6		273	16308094.34
2	-> Vector Streaming (type: GATHER)	6		273	16308094.34
3	-> Vector Hash Aggregate	6	16MB	273	16308092.67
4	-> Vector Hash Join (5,20)	6	585MB	169	16308092.63
5	-> Vector Streaming(type: REDISTRIBUTE)	1320811	1MB	181	16069870.93
6	-> Vector Hash Join (7,19)	1320811	43MB	181	16061891.00
7	-> Vector Streaming(type: REDISTRIBUTE)	1320811	1MB	131	16056566.78
8	-> Vector Hash Join (9,18)	1320811	27MB	131	16048586.85
9	-> Vector Streaming(type: REDISTRIBUTE)	1383248	1MB	131	16038321.62
10	-> Vector Hash Join (11,17)	1383248	16MB	131	16029664.50
11	-> Vector Hash Join (12,16)	2626366951	16MB	73	15751384.88
12	-> Vector Hash Join (13,14)	2750085660	2156MB	77	14226077.19
13	-> CStore Scan on store	24048	1MB	19	2264.00
14	-> Vector Partition Iterator	2879987999	1MB	66	2756066.50
15	-> Partitioned CStore Scan on store_sales	2879987999	1MB	66	2756066.50
16	-> CStore Scan on promotion	36000	1MB	4	1268.50
17	-> CStore Scan on item	158	1MB	58	4051.25
18	-> CStore Scan on customer	12000000	1MB	8	12923.00
19	-> CStore Scan on customer_address ad2	6000000	1MB	58	5770.00
20	-> Vector Partition Iterator	287999764	1MB	12	227383.99
21	-> Partitioned CStore Scan on store_returns	287999764	1MB	12	227383.99

(21 rows)

For details about the warning at the top of the plan, see [Hint Errors, Conflicts, and Other Warnings](#).

6.9.3 Join Operation Hints

Description

Specifies the join method, which can be nested-loop join, hash join, or merge join.

Syntax

```
[no] nestloop[hashjoin|mergejoin(table_list)]
```

Parameters

- 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 [Example](#) 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-memory	E-width	E-costs
1	-> Row Adapter	6		273	100061693161.06
2	-> Vector Streaming (type: GATHER)	6		273	100061693161.06
3	-> Vector Hash Aggregate	6	16MB	273	100061693159.40
4	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	169	100061693159.36
5	-> Vector Hash Join (6,22)	6	43MB	169	100061693158.99
6	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	119	100061688591.48
7	-> Vector Hash Join (8,21)	6	16MB	119	100061688591.30
8	-> Vector Hash Join (9,20)	7	27MB	123	100061687304.04
9	-> Vector Streaming(type: REDISTRIBUTE)	7	1MB	123	100061677823.27
10	-> Vector Hash Join (11,19)	7	16MB	123	100061677823.12
11	-> Vector Nest Loop (12,17)	7	1MB	112	100061675546.57
12	-> Vector Hash Join (13,15)	13670	585MB	62	6163443.54
13	-> Vector Partition Iterator	2879987999	1MB	66	2756066.50
14	-> Partitioned CStore Scan on store_sales	2879987999	1MB	66	2756066.50
15	-> Vector Partition Iterator	287999764	1MB	12	227383.99
16	-> Partitioned CStore Scan on store_returns	287999764	1MB	12	227383.99
17	-> Vector Materialize	158	16MB	58	4051.28
18	-> CStore Scan on item	158	1MB	58	4051.25
19	-> CStore Scan on store	24048	1MB	19	2264.00
20	-> CStore Scan on customer	12000000	1MB	8	12923.00
21	-> CStore Scan on promotion	36000	1MB	4	1268.50
22	-> CStore Scan on customer_address ad2	6000000	1MB	58	5770.00

(22 rows)

6.9.4 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(table_list #|+|-|* const)
```

Parameters

- #**, **+**, **-**, 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): Multiply the result set of joined **t1**, **t2**, and **t3** 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 [Example](#) 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-memory	E-width	E-costs
1	-> Row Adapter	312		273	3401656.58
2	-> Vector Streaming (type: GATHER)	312		273	3401656.58
3	-> Vector Hash Aggregate	312	16MB	273	3401634.91
4	-> Vector Streaming(type: REDISTRIBUTE)	313	1MB	169	3401634.39
5	-> Vector Hash Join (6,21)	313	43MB	169	3401633.06
6	-> Vector Streaming(type: REDISTRIBUTE)	313	1MB	119	3397065.38
7	-> Vector Hash Join (8,20)	313	27MB	119	3397064.31
8	-> Vector Streaming(type: REDISTRIBUTE)	328	1MB	119	3387583.37
9	-> Vector Hash Join (10,19)	328	16MB	119	3387582.18
10	-> Vector Hash Join (11,18)	344	16MB	123	3386294.74
11	-> Vector Hash Join (12,14)	360	19MB	112	3384018.02
12	-> Vector Partition Iterator	287999764	1MB	12	227383.99
13	-> Partitioned CStore Scan on store_returns	287999764	1MB	12	227383.99
14	-> Vector Hash Join (15,17)	1516824	16MB	124	3065686.08
15	-> Vector Partition Iterator	2879987999	1MB	66	2756066.50
16	-> Partitioned CStore Scan on store_sales	2879987999	1MB	66	2756066.50
17	-> CStore Scan on item	158	1MB	58	4051.25
18	-> CStore Scan on store	24048	1MB	19	2264.00
19	-> CStore Scan on promotion	36000	1MB	4	1268.50
20	-> CStore Scan on customer	12000000	1MB	8	12923.00
21	-> CStore Scan on customer_address ad2	6000000	1MB	58	5770.00

(21 rows)

The estimation value after the hint in row 11 is **360**, and the original value is rounded off to 7.

6.9.5 Stream Operation Hints

Description

These hints specify a stream operation, which can be **broadcast** or **redistribute**. You can also directly specify a method to generate a gather plan.

Syntax

```
[no] broadcast|redistribute(table_list)  
gather(REL|JOIN|ALL)
```

Parameters

- **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 [Parameters](#).
- **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 distributed 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 [Example](#) 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-memory	E-width	E-costs
1	-> Row Adapter	6		273	5718448.94
2	-> Vector Streaming (type: GATHER)	6		273	5718448.94
3	-> Vector Hash Aggregate	6	16MB	273	5718447.27
4	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	169	5718447.23
5	-> Vector Hash Join (6,21)	6	16MB	169	5718446.86
6	-> Vector Hash Join (7,20)	7	43MB	173	5717159.60
7	-> Vector Streaming(type: REDISTRIBUTE)	7	1MB	123	5712592.09
8	-> Vector Hash Join (9,18)	7	585MB	123	5712591.93
9	-> Vector Hash Join (10,17)	7	16MB	123	3386294.56
10	-> Vector Hash Join (11,13)	7	19MB	112	3384018.02
11	-> Vector Partition Iterator	287999764	1MB	12	227383.99
12	-> Partitioned CStore Scan on store_returns	287999764	1MB	12	227383.99
13	-> Vector Hash Join (14,16)	1516824	16MB	124	3065686.08
14	-> Vector Partition Iterator	2879987999	1MB	66	2756066.50
15	-> Partitioned CStore Scan on store_sales	2879987999	1MB	66	2756066.50
16	-> CStore Scan on item	158	1MB	58	4051.25
17	-> CStore Scan on store	24048	1MB	19	2264.00
18	-> Vector Streaming(type: BROADCAST)	288000000	1MB	8	2176297.36
19	-> CStore Scan on customer	12000000	1MB	8	12923.00
20	-> CStore Scan on customer_address ad2	6000000	1MB	58	5770.00
21	-> CStore Scan on promotion	36000	1MB	4	1268.50

(21 rows)

Specify the gather hint for a statement.

1. Generate the gather plan **/*+ GATHER(REL)*/** based on the base table.

openGauss=# 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.

openGauss=# 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.

```
openGauss=# 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)

6.9.6 Scan Operation Hints

Description

These hints specify a scan operation, which can be **tablescan**, **indexscan**, or **indexonlyscan**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

Parameters

- **no** indicates that the specified hint will not be used for a join.
- *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** or **indexonlyscan**. You can specify only one index.

NOTE

indexscan and **indexonlyscan** hints can be used only when the specified index belongs to the table.

Scan operation hints can be used for row-store tables and subquery tables.

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 [Example](#) 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:

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	6		273	100061674938.26
2	-> Vector Streaming (type: GATHER)	6		273	100061674938.26
3	-> Vector Hash Aggregate	6	16MB	273	100061674936.59
4	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	169	100061674936.55
5	-> Vector Hash Join (6,21)	6	43MB	169	100061674936.19
6	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	119	100061670368.67
7	-> Vector Hash Join (8,20)	6	16MB	119	100061670368.50
8	-> Vector Hash Join (9,19)	7	27MB	123	100061669081.23
9	-> Vector Streaming(type: REDISTRIBUTE)	7	1MB	123	100061659600.47
10	-> Vector Hash Join (11,18)	7	16MB	123	100061659600.31
11	-> Vector Nest Loop (12,17)	7	1MB	112	100061657323.77
12	-> Vector Hash Join (13,15)	13670	585MB	62	6163443.54
13	-> Vector Partition Iterator	2879987999	1MB	66	2756066.50
14	-> Partitioned CStore Scan on store_sales	2879987999	1MB	66	2756066.50
15	-> Vector Partition Iterator	287999764	1MB	12	227383.99
16	-> Partitioned CStore Scan on store_returns	287999764	1MB	12	227383.99
17	-> CStore Index Scan using i on item	1	1MB	58	4.01
18	-> CStore Scan on store	24048	1MB	19	2264.00
19	-> CStore Scan on customer	12000000	1MB	8	12923.00
20	-> CStore Scan on promotion	36000	1MB	4	1268.50
21	-> CStore Scan on customer_address ad2	6000000	1MB	58	5770.00

(21 rows)

6.9.7 Sublink Name Hints

Description

These hints specify the name of a sublink block.

Syntax

blockname (table)

Parameters

- 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.

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	1439994000		216	325105765847.91
2	-> Vector Streaming (type: GATHER)	1439994000		216	325105765847.91
3	-> Vector Nest Loop Semi Join (4, 6)	1439994000	1MB	216	325026664615.00
4	-> Vector Partition Iterator	2879987999	1MB	216	2756066.50
5	-> Partitioned CStore Scan on store_sales	2879987999	1MB	216	2756066.50
6	-> Vector Materialize	300000	16MB	4	4176.25
7	-> Vector Hash Aggregate	300000	16MB	4	3988.75
8	-> CStore Scan on item	300000	1MB	4	3832.50

(8 rows)

6.9.8 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(table (column) [(value)])`
- Specify intermediate result skew.
`skew((join_rel) (column) [(value)])`

Parameters

- 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.

Example

Specify single-table skew.

- Specify hints in the original query.

For example, the original query is as follows:

```
explain
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
select c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	100		20	911254.47
2	-> Vector Limit	100		20	911254.47
3	-> Vector Streaming (type: GATHER)	2400		20	911325.75
4	-> Vector Limit	2400	1MB	20	911247.62
5	-> Vector Sort	3684816	16MB	20	911631.21
6	-> Vector Hash Join (7,29)	3684817	41MB (12374MB)	20	905379.41
7	-> Vector Streaming(type: REDISTRIBUTE)	3684817	384KB	4	883010.31
8	-> Vector Hash Join (9,19)	3684817	16MB	4	861302.05
9	-> Vector Hash Join (10,18)	11054450	16MB	44	427109.71
10	-> Vector Hash Aggregate	50247501	397MB (12671MB)	54	395302.57
11	-> Vector Streaming(type: REDISTRIBUTE)	50247501	384KB	22	358663.76
12	-> Vector Hash Join (13,15)	50247501	16MB	22	294300.51
13	-> Vector Partition Iterator	287999764	1MB	26	227383.99
14	-> Partitioned CStore Scan on store_returns	287999764	1MB	26	227383.99
15	-> Vector Streaming(type: BROADCAST)	8712	384KB	4	975.56
16	-> Vector Partition Iterator	363	1MB	4	910.65
17	-> Partitioned CStore Scan on date_dim	363	1MB	4	910.65
18	-> CStore Scan on store	44	1MB	4	1006.39
19	-> Vector Hash Aggregate	192	16MB	68	426707.38
20	-> Vector Subquery Scan on ctr2	50247501	1MB	36	416239.03
21	-> Vector Hash Aggregate	50247501	397MB (12671MB)	54	395302.57
22	-> Vector Streaming(type: REDISTRIBUTE)	50247501	384KB	22	358663.76
23	-> Vector Hash Join (24,26)	50247501	16MB	22	294300.51
24	-> Vector Partition Iterator	287999764	1MB	26	227383.99
25	-> Partitioned CStore Scan on store_returns	287999764	1MB	26	227383.99
26	-> Vector Streaming(type: BROADCAST)	8712	384KB	4	975.56
27	-> Vector Partition Iterator	363	1MB	4	910.65
28	-> Partitioned CStore Scan on date_dim	363	1MB	4	910.65
29	-> CStore Scan on customer	12000000	1MB	24	12923.00
(29 rows)					

Specify the hints of HashAgg in the inner **with** clause and of the outer Hash Join. The query containing hints is as follows:

```
explain
with customer_total_return as
(select /*+ skew(store_returns(sr_store_sk sr_customer_sk)) */sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
select /*+ skew(ctr1(ctr_customer_sk)(11))*/ c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
```

```
order by c_customer_id  
limit 100;
```

The hints indicate that the **group by** in the inner **with** clause contains skew data during redistribution by HashAgg, corresponding to the original Hash Agg operators 10 and 21; and that the **ctr_customer_sk** column in the outer **ctr1** table contains skew data during redistribution by Hash Join, corresponding to operator 6 in the original plan. The optimized plan is as follows:

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	100		20	1061778.14
2	-> Vector Limit	100		20	1061778.14
3	-> Vector Streaming (type: GATHER)	2400		20	1061849.41
4	-> Vector Limit	2400	1MB	20	1061771.29
5	-> Vector Sort	3684816	16MB	20	1062154.87
6	-> Vector Hash Join (7,31)	3684817	41MB (12344MB)	20	1055903.08
7	-> Vector Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)	3684817	384KB	4	1013056.49
8	-> Vector Hash Join (9,20)	3684817	16MB	4	1000066.10
9	-> Vector Hash Join (10,19)	11054450	16MB	44	496461.73
10	-> Vector Hash Aggregate	50247501	397MB (12010MB)	54	464654.59
11	-> Vector Streaming(type: REDISTRIBUTE)	50247501	384KB	54	428015.79
12	-> Vector Hash Aggregate	50247501	397MB (12010MB)	54	330939.31
13	-> Vector Hash Join (14,16)	50247501	16MB	22	284900.51
14	-> Vector Partition Iterator	287999764	1MB	26	227383.99
15	-> Partitioned CStore Scan on store_returns	287999764	1MB	26	227383.99
16	-> Vector Streaming(type: BROADCAST)	8712	384KB	4	975.56
17	-> Vector Partition Iterator	363	1MB	4	910.65
18	-> Partitioned CStore Scan on date_dim	363	1MB	4	910.65
19	-> CStore Scan on store	44	1MB	4	1006.39
20	-> Vector Hash Aggregate	192	16MB	68	496059.40
21	-> Vector Subquery Scan on ctr2	50247501	1MB	36	485591.05
22	-> Vector Hash Aggregate	50247501	397MB (12010MB)	54	464654.59
23	-> Vector Streaming(type: REDISTRIBUTE)	50247501	384KB	54	428015.79
24	-> Vector Hash Aggregate	50247501	397MB (12010MB)	54	330939.31
25	-> Vector Hash Join (26,28)	50247501	16MB	22	284900.51
26	-> Vector Partition Iterator	287999764	1MB	26	227383.99
27	-> Partitioned CStore Scan on store_returns	287999764	1MB	26	227383.99
28	-> Vector Streaming(type: BROADCAST)	8712	384KB	4	975.56
29	-> Vector Partition Iterator	363	1MB	4	910.65
30	-> Partitioned CStore Scan on date_dim	363	1MB	4	910.65
31	-> Vector Streaming(type: PART LOCAL PART BROADCAST)	12000000	384KB	24	34685.50
32	-> CStore Scan on customer	12000000	1MB	24	12923.00

(32 rows)

To solve data skew in the redistribution, Hash Agg is changed to double-level Agg operators and the redistribution operators used by Hash Join are changed in the optimized plan.

- Modify the query and then specify hints.

For example, the original query and its plan are as follows:

```
explain select count(*) from store_sales_1 group by round(ss_list_price);
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	16672		14	62261.28
2	-> Vector Streaming (type: GATHER)	16672		14	62261.28
3	-> Vector Streaming(type: LOCAL GATHER dop: 1/2)	16672	32KB	14	61479.78
4	-> Vector Hash Aggregate	16672	16MB	14	61452.00
5	-> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)	3112836	128KB	6	57498.43
6	-> CStore Scan on store_sales_1	3112836	1MB	6	21810.25

(6 rows)

Columns in hints do not support expressions. To specify hints, rewrite the query as several subqueries. The rewritten query and its plan are as follows:

```
explain  
select count(*)  
from (select round(ss_list_price),ss_hdemo_sk  
from store_sales_1)tmp(a,ss_hdemo_sk)  
group by a;
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	16672		14	62261.28
2	-> Vector Streaming (type: GATHER)	16672		14	62261.28
3	-> Vector Streaming(type: LOCAL GATHER dop: 1/2)	16672	32KB	14	61479.78
4	-> Vector Hash Aggregate	16672	16MB	14	61452.00
5	-> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)	3112836	128KB	6	57498.43
6	-> CStore Scan on store_sales_1	3112836	1MB	6	21810.25

(6 rows)

Ensure that the service logic is not changed during the rewriting.

Specify hints in the rewritten query as follows:

```
explain  
select /*+ skew(tmp(a)) */ count(*)  
from (select round(ss_list_price),ss_hdemo_sk  
from store_sales_1)tmp(a,ss_hdemo_sk)  
group by a;
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	16672		14	27771.82
2	-> Vector Streaming (type: GATHER)	16672		14	27771.82
3	-> Vector Streaming (type: LOCAL GATHER dop: 1/2)	16672	32KB	14	26990.32
4	-> Vector Hash Aggregate	16671	16MB	14	26962.54
5	-> Vector Streaming (type: SPLIT REDISTRIBUTE dop: 2/2)	66216	128KB	14	26838.09
6	-> Vector Hash Aggregate	66216	16MB	14	25949.61
7	-> CStore Scan on store_sales_1	3112836	1MB	6	21810.25

(7 rows)

The plan shows that after Hash Agg is changed to double-layer Agg operators, redistributed data is greatly reduced and redistribution time is shortened.

You can specify hints in columns in a subquery, for example:

```
explain
select /*+ skew(tmp(b)) */ count(*)
from (select round(ss_list_price) b,ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

6.9.9 Parameterized Path Hint

Description

Specifies the parameterized path and the conditional predicate pushdown method.

Syntax

```
predpush(src1 src2)
predpush(src, dest)
```

Parameters

- **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:

```
set rewrite_rule = 'predpushnormal';
explain (costs off) SELECT /*+PREDPUSH(t2, st3)*/ *
FROM t2,
      (SELECT sum(t3.b), t3.a FROM t3, t4 where t3.a = t4.a GROUP BY t3.a) st3
WHERE st3.a = t2.a;
id | operation
---+-----
1 | -> Streaming (type: GATHER)
2 | -> Nested Loop (3,4)
```

```
3 | -> Seq Scan on t2
4 | -> GroupAggregate
5 |   -> Nested Loop (6,7)
6 |     -> Index Only Scan using t4_a_idx on t4
7 |       -> Materialize
8 |         -> Index Scan using t3_a_idx on t3
(8 rows)

Predicate Information (identified by plan id)
-----
 6 --Index Only Scan using t4_a_idx on t4
   Index Cond: (a = t2.a)
 8 --Index Scan using t3_a_idx on t3
   Index Cond: (a = t2.a)
(4 rows)
```

If the **predpush** hint is not used, t3 and t4 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 t3 and t4 based on t2 before they are joined. The result set returned after joining is small, which effectively improves the performance.

6.9.10 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 **hashjoin** or **mergejoin** is specified for non-equivalent joins, it will not be used.
- If **indexscan** or **indexonlyscan** is specified for a table that does not have an index, it will not be used.
- If **indexscan hint** or **indexonlyscan** is specified for a full-table scan or for a scan whose filtering conditions are not set on index columns, it will not be used.
- The specified **indexonlyscan** hint is used only when the output column contains only indexes.
- 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.

6.9.11 Plan Hint Cases

This section takes the statements in TPC-DS (Q24) as an example to describe how to optimize an execution plan by using hints in an environment with more than 1000 data sets and 24 DNs. For example:

```
select avg(netpaid) from
(select c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size
,sum(ss_sales_price) netpaid
from store_sales
,store_returns
,store
,item
,customer
,customer_address
where ss_ticket_number = sr_ticket_number
and ss_item_sk = sr_item_sk
and ss_customer_sk = c_customer_sk
and ss_item_sk = i_item_sk
and ss_store_sk = s_store_sk
and c_birth_country = upper(ca_country)
and s_zip = ca_zip
and s_market_id=7
group by c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size);
```

1. The original plan of this statement is as follows and the statement execution takes 110s:

id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	110324.107	1	1
2	-> Vector Aggregate	110324.093	1	1
3	-> Vector Streaming (type: GATHER)	110323.958	24	24
4	-> Vector Aggregate	[110179.302,110309.653]	24	24
5	-> Vector Hash Aggregate	[110178.388,110308.515]	647824	16656
6	-> Vector Streaming(type: REDISTRIBUTE)	[77616.177,96478.771]	666834733	16664
7	-> Vector Hash Join (8,22)	[81727.257,84728.519]	666834733	16664
8	-> Vector Streaming(type: REDISTRIBUTE)	[78770.520,82021.087]	666834733	16664
9	-> Vector Hash Join (10,21)	[88066.755,90701.860]	666834733	16664
10	-> Vector Streaming(type: BROADCAST)	[7940.962,21430.725]	591882336	51360
11	-> Vector Hash Join (12,20)	[2419.995,5319.606]	24661764	2140
12	-> Vector Streaming(type: REDISTRIBUTE)	[1750.448,4659.581]	25258268	2241
13	-> Vector Hash Join (14,18)	[15240.666,17159.616]	25258268	2241
14	-> Vector Hash Join (15,17)	[12112.913,13563.366]	252564412	472070592
15	-> Vector Partition Iterator	[11148.731,12473.230]	2879987999	2879987999
16	-> Partitioned CStore Scan on public.store_sales	[11097.921,12412.596]	2879987999	2879987999
17	-> CStore Scan on public.store	[0.447,0.689]	2064	2064
18	-> Vector Partition Iterator	[296.805,319.014]	287999764	287999764
19	-> Partitioned CStore Scan on public.store_returns	[292.938,314.787]	287999764	287999764
20	-> CStore Scan on public.customer	[114.358,144.462]	12000000	12000000
21	-> CStore Scan on public.customer_address	[38.426,56.753]	6000000	6000000
22	-> CStore Scan on public.item	[3.160,5.026]	300000	300000

(22 rows)

In this plan, the performance of the layer-10 **broadcast** is poor because the estimation result generated at layer 11 is 2140 rows, much less than the actual number of rows. The inaccurate estimation is mainly caused by the underestimated number of rows in layer-13 hash join. In this layer, **store_sales** and **store_returns** are joined (based on the **ss_ticket_number** and **ss_item_sk** columns in **store_sales** and the **sr_ticket_number** and **sr_item_sk** columns in **store_returns**) but the multi-column join is not considered.

2. After the **rows** hint is used for optimization, the plan is as follows and the statement execution takes 318s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns * 11270)*/ c_last_name ...
```

id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	318585.246	1	1
2	-> Vector Aggregate	318585.232	1	1
3	-> Vector Streaming (type: GATHER)	318585.082	24	24
4	-> Vector Aggregate	[318323.324,318499.290]	24	24
5	-> Vector Hash Aggregate	[318320.813,318497.054]	647824	187770504
6	-> Vector Streaming(type: REDISTRIBUTE)	[288074.860,305601.698]	666834733	187770507
7	-> Vector Hash Join (8,22)	[253642.468,315808.664]	666834733	187770507
8	-> Vector Hash Join (9,18)	[250904.317,315684.018]	666834733	187770507
9	-> Vector Streaming(type: REDISTRIBUTE)	[4552.500,310602.307]	275042158	147106999
10	-> Vector Hash Join (11,17)	[7658.951,14053.823]	275042158	147106999
11	-> Vector Streaming(type: REDISTRIBUTE)	[3953.255,10264.943]	287999764	154060900
12	-> Vector Hash Join (13,15)	[28196.188,32838.794]	287999764	154060900
13	-> Vector Partition Iterator	[11477.673,12324.583]	2879987999	2879987999
14	-> Partitioned CStore Scan on public.store_sales	[11411.382,12250.209]	2879987999	2879987999
15	-> Vector Partition Iterator	[304.188,403.205]	287999764	287999764
16	-> Partitioned CStore Scan on public.store_returns	[299.838,398.255]	287999764	287999764
17	-> CStore Scan on public.customer	[122.246,170.128]	12000000	12000000
18	-> Vector Streaming(type: REDISTRIBUTE)	[57.558,117.461]	492915	146467
19	-> Vector Hash Join (20,21)	[45.554,96.238]	492915	146467
20	-> CStore Scan on public.customer_address	[39.738,89.412]	6000000	6000000
21	-> CStore Scan on public.store	[0.361,1.095]	2064	2064
22	-> Vector Streaming(type: BROADCAST)	[48.986,91.170]	7200000	7200000
23	-> CStore Scan on public.item	[4.506,6.602]	300000	300000

(23 rows)

The execution takes a longer time because layer-9 **redistribute** is slow. Considering that data skew does not occur at layer-9 **redistribute**, the slow redistribution is caused by the slow layer-8 **hashjoin** due to data skew at layer-18 **redistribute**.

3. Data skew occurs at layer-18 **redistribute** because **customer_address** has a few different values in its two join keys. Therefore, plan **customer_address** as the last one to be joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 116s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((store_sales store_returns store item customer) customer_address)*/
c_last_name ...
```


id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	116326.597	1	1
2	-> Vector Aggregate	116326.590	1	1
3	-> Vector Streaming (type: GATHER)	116326.473	24	24
4	-> Vector Aggregate	[116157.161,116236.494]	24	24
5	-> Vector Hash Aggregate	[116155.328,116233.946]	647824	187770504
6	-> Vector Streaming(type: REDISTRIBUTE)	[84103.951,102052.326]	666834733	187770507
7	-> Vector Hash Join (8,10)	[23229.469,47484.697]	666834733	187770507
8	-> Vector Streaming(type: REDISTRIBUTE)	[38.367,74.930]	6000000	6000000
9	-> CStore Scan on public.customer_address	[69.877,121.460]	6000000	6000000
10	-> Vector Streaming(type: REDISTRIBUTE)	[17404.744,17567.550]	24661764	24112909
11	-> Vector Hash Join (12,22)	[16123.627,16397.246]	24661764	24112909
12	-> Vector Streaming(type: REDISTRIBUTE)	[15320.663,15741.646]	25258268	25252751
13	-> Vector Hash Join (14,21)	[14962.342,16375.458]	25258268	25252751
14	-> Vector Hash Join (15,19)	[14449.031,15825.949]	25258268	25252751
15	-> Vector Hash Join (16,18)	[11439.959,12510.065]	252564412	472070592
16	-> Vector Partition Iterator	[10531.986,11536.213]	2879987999	2879987999
17	-> Partitioned CStore Scan on public.store_sales	[10483.634,11474.944]	2879987999	2879987999
18	-> CStore Scan on public.store	[0.347,0.463]	2064	2064
19	-> Vector Partition Iterator	[293.977,365.021]	287999764	287999764
20	-> Partitioned CStore Scan on public.store_returns	[289.936,360.808]	287999764	287999764
21	-> CStore Scan on public.item	[3.109,5.245]	300000	300000
22	-> CStore Scan on public.customer	[113.871,141.791]	12000000	12000000
(22 rows)				

Most of the time is spent on layer-6 **redistribute**. The plan needs to be further optimized.

4. Most of the time is spent on layer-6 **redistribute** because of data skew. To avoid the data skew, plan the **item** table as the last one to be joined because the number of rows is not reduced after **item** is joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 120s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
c_last_name ...
```

id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	120377.258	1	1
2	-> Vector Aggregate	120377.245	1	1
3	-> Vector Streaming (type: GATHER)	120377.091	24	24
4	-> Vector Aggregate	[120184.884,120301.704]	24	24
5	-> Vector Hash Aggregate	[120183.119,120297.845]	647824	187770504
6	-> Vector Streaming(type: REDISTRIBUTE)	[87775.682,106070.878]	666834733	187770507
7	-> Vector Hash Join (8,22)	[22323.764,49878.523]	666834733	187770507
8	-> Vector Hash Join (9,11)	[21129.236,45208.255]	666834733	187770507
9	-> Vector Streaming(type: REDISTRIBUTE)	[37.859,75.412]	6000000	6000000
10	-> CStore Scan on public.customer_address	[74.798,114.449]	6000000	6000000
11	-> Vector Streaming(type: REDISTRIBUTE)	[15714.458,15824.928]	24661764	24112909
12	-> Vector Hash Join (13,21)	[14637.516,14955.464]	24661764	24112909
13	-> Vector Streaming(type: REDISTRIBUTE)	[13898.593,14333.200]	25258268	25252751
14	-> Vector Hash Join (15,19)	[14166.917,15378.244]	25258268	25252751
15	-> Vector Hash Join (16,18)	[11272.239,12052.532]	252564412	472070592
16	-> Vector Partition Iterator	[10409.566,11127.981]	2879987999	2879987999
17	-> Partitioned CStore Scan on public.store_sales	[10365.838,11077.601]	2879987999	2879987999
18	-> CStore Scan on public.store	[0.431,0.609]	2064	2064
19	-> Vector Partition Iterator	[343.780,408.254]	287999764	287999764
20	-> Partitioned CStore Scan on public.store_returns	[339.844,403.923]	287999764	287999764
21	-> CStore Scan on public.customer	[117.234,163.598]	12000000	12000000
22	-> Vector Streaming(type: BROADCAST)	[44.571,130.129]	7200000	7200000
23	-> CStore Scan on public.item	[4.169,6.347]	300000	300000
(23 rows)				

Data skew occurs after the join of **item** and **customer_address** because **item** is broadcast at layer-22. As a result, layer-6 **redistribute** is still slow.

5. Add a hint to disable **broadcast** for **item** or add a **redistribute** hint for the join result of **item** and **customer_address**. After the hint is used for optimization, the plan is as follows and the statement execution takes 105s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
no broadcast(item)*
c_last_name ...
```

id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	105854.957	1	1
2	-> Vector Aggregate	105854.948	1	1
3	-> Vector Streaming (type: GATHER)	105854.825	24	24
4	-> Vector Aggregate	[105706.709,105776.135]	24	24
5	-> Vector Hash Aggregate	[105705.061,105773.013]	647824	187770504
6	-> Vector Streaming(type: REDISTRIBUTE)	[70701.966,89973.672]	666834733	187770507
7	-> Vector Hash Join (8,23)	[71759.500,79018.433]	666834733	187770507
8	-> Vector Streaming(type: REDISTRIBUTE)	[69794.307,77269.178]	666834733	187770507
9	-> Vector Hash Join (10,12)	[21443.307,46714.378]	666834733	187770507
10	-> Vector Streaming(type: REDISTRIBUTE)	[41.295,83.419]	6000000	6000000
11	-> CStore Scan on public.customer_address	[70.405,166.072]	6000000	6000000
12	-> Vector Streaming(type: REDISTRIBUTE)	[15689.053,15788.475]	24661764	24112909
13	-> Vector Hash Join (14,22)	[14517.847,14712.929]	24661764	24112909
14	-> Vector Streaming(type: REDISTRIBUTE)	[13806.733,14089.770]	25258268	25252751
15	-> Vector Hash Join (16,20)	[13709.384,15095.449]	25258268	25252751
16	-> Vector Hash Join (17,19)	[10944.796,11827.285]	252564412	472070592
17	-> Vector Partition Iterator	[10070.316,10884.728]	2879987999	2879987999
18	-> Partitioned CStore Scan on public.store_sales	[10018.966,10828.990]	2879987999	2879987999
19	-> CStore Scan on public.store	[0.447,0.568]	2064	2064
20	-> Vector Partition Iterator	[293.042,329.056]	287999764	287999764
21	-> Partitioned CStore Scan on public.store_returns	[288.631,324.782]	287999764	287999764
22	-> CStore Scan on public.customer	[113.735,138.235]	12000000	12000000
23	-> CStore Scan on public.item	[3.127,5.357]	300000	300000

(23 rows)

6. The last layer uses single-layer **Agg** and the number of rows is greatly reduced. Set **best_agg_plan** to **3** and change the single-layer **Agg** to a double-layer **Agg**. The plan is as follows and the statement execution takes 94s. The optimization ends.

id	operation	A-time	A-rows	E-rows
1	-> Row Adapter	94004.670	1	1
2	-> Vector Aggregate	94004.655	1	1
3	-> Vector Streaming (type: GATHER)	94004.504	24	24
4	-> Vector Aggregate	[93833.832,93928.052]	24	24
5	-> Vector Hash Aggregate	[93832.460,93926.412]	647824	187770507
6	-> Vector Streaming(type: REDISTRIBUTE)	[93640.866,93787.939]	647824	183912384
7	-> Vector Hash Aggregate	[93687.544,93791.242]	647824	183912384
8	-> Vector Hash Join (9,24)	[70025.469,72773.161]	666834733	187770507
9	-> Vector Streaming(type: REDISTRIBUTE)	[68242.223,71275.972]	666834733	187770507
10	-> Vector Hash Join (11,13)	[21421.136,44830.306]	666834733	187770507
11	-> Vector Streaming(type: REDISTRIBUTE)	[35.444,71.328]	6000000	6000000
12	-> CStore Scan on public.customer_address	[67.246,119.224]	6000000	6000000
13	-> Vector Streaming(type: REDISTRIBUTE)	[16089.853,16212.570]	24661764	24112909
14	-> Vector Hash Join (15,23)	[14822.972,15188.942]	24661764	24112909
15	-> Vector Streaming(type: REDISTRIBUTE)	[14061.867,14604.162]	25258268	25252751
16	-> Vector Hash Join (17,21)	[13949.756,15492.311]	25258268	25252751
17	-> Vector Hash Join (18,20)	[10935.742,12160.719]	252564412	472070592
18	-> Vector Partition Iterator	[10052.958,11194.962]	2879987999	2879987999
19	-> Partitioned CStore Scan on public.store_sales	[10008.415,11143.984]	2879987999	2879987999
20	-> CStore Scan on public.store	[0.452,0.839]	2064	2064
21	-> Vector Partition Iterator	[298.235,332.736]	287999764	287999764
22	-> Partitioned CStore Scan on public.store_returns	[294.067,327.629]	287999764	287999764
23	-> CStore Scan on public.customer	[114.377,145.156]	12000000	12000000
24	-> CStore Scan on public.item	[3.150,5.530]	300000	300000

(24 rows)

If the query performance deteriorates due to statistics changes, you can use hints to optimize the query plan. Take TPC-H-Q17 as an example. The query performance deteriorates after the value of **default_statistics_target** is changed from the default one to **-2** for statistics collection.

1. If **default_statistics_target** is set to the default value **100**, the plan is as follows:

id	operation	A-time
1	-> Row Adapter	265006.779
2	-> Vector Aggregate	265006.764
3	-> Vector Streaming (type: GATHER)	265006.071
4	-> Vector Aggregate	[263699.512,264503.084]
5	-> Vector Hash Join (6,17)	[263676.665,264477.932]
6	-> Vector Streaming(type: LOCAL GATHER dop: 1/4)	[1.998,7.594]
7	-> Vector Hash Aggregate	[201775.393,202432.672]
8	-> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)	[201567.130,202231.524]
9	-> Vector Hash Join (10,12)	[170675.231,199908.410]
10	-> Vector Partition Iterator	[34847.797,51968.266]
11	-> Partitioned CStore Scan on tpch10wx_col.lineitem	[33805.013,51137.657]
12	-> Vector Hash Aggregate	[23283.387,25359.493]
13	-> Vector Streaming(type: SPLIT BROADCAST dop: 4/4)	[12850.624,14608.515]
14	-> Vector Hash Aggregate	[2690.439,3616.623]
15	-> Vector Partition Iterator	[2659.700,3579.390]
16	-> Partitioned CStore Scan on tpch10wx_col.part	[2642.213,3559.093]
17	-> Vector Streaming(type: REDISTRIBUTE dop: 1/4)	[262300.732,262961.078]
18	-> Vector Hash Join (19,21)	[225749.727,260990.322]
19	-> Vector Partition Iterator	[40046.078,56220.694]
20	-> Partitioned CStore Scan on tpch10wx_col.lineitem	[39204.414,55328.448]
21	-> Vector Streaming(type: SPLIT BROADCAST dop: 4/4)	[55748.177,61987.136]
22	-> Vector Partition Iterator	[3042.864,3873.942]
23	-> Partitioned CStore Scan on tpch10wx_col.part	[3027.023,3848.159]

(23 rows)

2. If **default_statistics_target** is set to **-2**, the plan is as follows:

id	operation	A-time
1	-> Row Adapter	1440492.994
2	-> Vector Aggregate	1440492.982
3	-> Vector Streaming (type: GATHER)	1440491.021
4	-> Vector Streaming(type: LOCAL GATHER dop: 1/6)	[1439737.284,1440008.568]
5	-> Vector Aggregate	[1439008.369,1439854.148]
6	-> Vector Hash Join (7,18)	[1439006.016,1439851.619]
7	-> Vector Streaming(type: LOCAL BROADCAST dop: 6/6)	[2.932,139.405]
8	-> Vector Hash Aggregate	[190452.312,195910.748]
9	-> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)	[190171.929,195653.119]
10	-> Vector Hash Join (11,13)	[161076.195,178831.123]
11	-> Vector Partition Iterator	[27306.318,45564.565]
12	-> Partitioned CStore Scan on tpch10wx_col.lineitem	[26752.444,44912.020]
13	-> Vector Hash Aggregate	[35601.624,39812.058]
14	-> Vector Streaming(type: SPLIT BROADCAST dop: 6/6)	[23096.460,27057.137]
15	-> Vector Hash Aggregate	[2372.587,3052.445]
16	-> Vector Partition Iterator	[2345.381,3012.732]
17	-> Partitioned CStore Scan on tpch10wx_col.part	[2329.874,2989.393]
18	-> Vector Hash Join (19,22)	[1437388.414,1438470.781]
19	-> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)	[1392693.529,1408571.859]
20	-> Vector Partition Iterator	[29065.204,41264.514]
21	-> Partitioned CStore Scan on tpch10wx_col.lineitem	[28212.219,40133.491]
22	-> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 6/6)	[2570.841,3438.567]
23	-> Vector Partition Iterator	[2447.569,3276.369]
24	-> Partitioned CStore Scan on tpch10wx_col.part	[2432.124,3263.641]
(24 rows)		

3. After the analysis, the cause is that the stream type is changed from **Broadcast** to **Redistribute** during the join of the **lineitem** and **part** tables. You can use a hint to change the stream type back to **BroadCast**. For example:

```
select /*+ no redistribute(part lineitem) */
      sum(l_extendedprice) / 7.0 as avg_yearly
from
      lineitem,
      part
where
      p_partkey = l_partkey
      and p_brand = 'Brand#23'
      and p_container = 'MED BOX'
      and l_quantity < (
          select
              0.2 * avg(l_quantity)
            from
              lineitem
           where
              l_partkey = p_partkey
        );
```

6.9.12 Optimizer GUC Parameter Hints

Description

These hints set GUC parameters related to query optimization that take effect during the query execution. For details about the application scenarios of hints, see the description of each GUC parameter.

Syntax

```
set(param value)
```

Parameters

- **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
enable_bitmapscan, enable_hashagg, enable_hashjoin, enable_indexscan, enable_indexonlyscan, 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, and enable_remotesort
 - Integer type
best_agg_plan and query_dop
 - Floating point
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
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 joins.
 - 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.
- CREATE TABLE ... AS ... The GUC parameter hint in the outermost query takes effect.

6.9.13 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)
```

6.9.14 Hints Specifying Not to Expand Subqueries

Description

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 selectivity 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
```

Example

Normal query execution:

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

Plan:

```
QUERY PLAN  
-----  
Streaming (type: GATHER) (cost=0.06..2.13 rows=2 width=12)  
Node/s: All datanodes  
-> Nested Loop Semi Join (cost=0.00..2.03 rows=2 width=12)  
    Join Filter: (t1.a = t2.a)  
    -> Seq Scan on t1 (cost=0.00..1.01 rows=1 width=12)  
    -> Seq Scan on t2 (cost=0.00..1.01 rows=1 width=4)  
(6 rows)
```

After `no_expand` is added:

```
explain select * from t1 where t1.a in (select /*+ no_expand*/ t2.a from t2);
```

Plan:

```
----- QUERY PLAN -----
Streaming (type: GATHER) (cost=1.09..2.13 rows=1 width=12)
  Node/s: All datanodes
  -> Seq Scan on t1 (cost=1.02..2.04 rows=1 width=12)
      Filter: (hashed SubPlan 1)
      SubPlan 1
        -> Materialize (cost=0.00..1.02 rows=4 width=4)
            -> Streaming(type: BROADCAST) (cost=0.00..1.01 rows=2 width=4)
                Spawn on: All datanodes
                -> Seq Scan on t2 (cost=0.00..1.01 rows=1 width=4)

(9 rows)
```

6.9.15 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

```
openGauss=# deallocate all;
DEALLOCATE ALL
openGauss=# prepare insert_nogpc as insert /*+ no_gpc */ into t1 select c1, c2 from t2 where c1 = $1;
PREPARE
openGauss=# execute insert_nogpc(1);
INSERT 0 1
openGauss=# select * from db_perf.global_plancache_status where schema_name = 'schema_hint_iud' order by 1,2;
 node_name | query | refcount | valid | databaseid | schema_name | params_num | func_id
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

No result exists in the **db_perf.global_plancache_status** view, that is, no plan is cached globally.

6.9.16 Hint of Parameterized Paths at the Same Level

Description

The **predpush_same_level** hint is 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(src, dest)
predpush_same_level(src1 src2 ..., dest)
```

 NOTE

This parameter takes effect only when the **predpushforce** option in **rewrite_rule** is enabled.

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;
```

Prepare parameters, tables, and indexes.

```
openGauss=# set rewrite_rule = 'predpushforce';  
SET  
openGauss=# create table t1(a int, b int) distribute by hash(a);  
CREATE TABLE  
openGauss=# create table t2(a int, b int) distribute by hash(a);  
CREATE TABLE  
openGauss=# create index idx1 on t1(a);  
CREATE INDEX  
openGauss=# create index idx2 on t2(a);  
CREATE INDEX
```

Run the following statement to view the plan:

```
openGauss=# explain select * from t1, t2 where t1.a = t2.a;  
QUERY PLAN  
-----  
Streaming (type: GATHER) (cost=18.25..77.00 rows=1000 width=16)  
Node/s: All datanodes  
-> Hash Join (cost=14.25..30.12 rows=1000 width=16)  
    Hash Cond: (t1.a = t2.a)  
    -> Seq Scan on t1 (cost=0.00..9.00 rows=1000 width=8)  
    -> Hash (cost=8.00..8.00 rows=1000 width=8)  
        -> Seq Scan on t2 (cost=0.00..8.00 rows=1000 width=8)  
(7 rows)
```

The filter condition **t1.a = t2.a** 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.

```
openGauss=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.a = t2.a;  
QUERY PLAN  
-----  
Streaming (type: GATHER) (cost=4.00..263.88 rows=1000 width=16)  
Node/s: All datanodes  
-> Nested Loop (cost=0.00..217.00 rows=1000 width=16)  
    -> Seq Scan on t1 (cost=0.00..9.00 rows=1000 width=8)  
    -> Index Scan using idx2 on t2 (cost=0.00..0.41 rows=1 width=8)  
        Index Cond: (a = t1.a)  
(6 rows)
```

NOTICE

- **predpush_same_level** 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 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, GaussDB provides a GUC option: **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".  
Please check for potential performance problem.
```

 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.

6.11 Optimization Cases

6.11.1 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 DN's, as shown in [Figure 6-10](#).

Figure 6-10 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 6-11](#).

Figure 6-11 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)
```

6.11.2 Case: Creating an Appropriate Index

Symptom

Query the information about all personnel in the sales department.

```
-- Create tables.
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)
```

6.11.3 Case: Adding NOT NULL for JOIN Columns

Symptom

```
SELECT
*
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
```

```
SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
FROM
PS.SDR_WEB_BSCRNC_1DAY SDR
INNER JOIN (SELECT
  BSCRNC_ID,
  BSCRNC_NAME,
  ACCESS_TYPE,
  ACCESS_TYPE_ID
FROM
  nethouse.DIM_LOC_BSCRNC
GROUP BY
  BSCRNC_ID,
  BSCRNC_NAME,
  ACCESS_TYPE,
  ACCESS_TYPE_ID) DIM
ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
AND DIM.ACCESS_TYPE_ID IN (0,1,2)
INNER JOIN nethouse.DIM_RAT_MAPPING RAT
ON (RAT.RAT = SDR.RAT)
WHERE
( ( STARTTIME >= 1461340800
AND STARTTIME < 1461427200 )
AND RAT.ACCESS_TYPE_ID IN (0,1,2)
--and SDR.BSCRNC_ID is not null
GROUP BY
STTIME ) ;
```

Figure 6-12 shows the execution plan.

Figure 6-12 Adding NOT NULL for JOIN columns (1)

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Row Adapter	13055.792	1	72	72KB	1		160	1.00444120.99
2	Vector Streaming (type: GATHER)	13055.779	1	72	144KB	1		160	1.00444120.99
3	Vector Hash Aggregate	15621.925,3679.694	1	1	13004KB, 9000KB	1	[75, 78]	55	1.0648007.29
4	Vector Streaming (type: REDISTRIBUTE)	15621.909,3679.593	72	2	2404KB, 2404KB	1		55	1.0648007.32
5	Vector Hash Aggregate	13916.657,3634.515	72	2	13013KB, 5013KB	1	[75, 78]	55	1.0648007.29
6	Vector Hash Join (1,2)	13236.674,3545.293	1	3665620	1.094077	1	[17784KB, 51411KB]	1405	1.064123.67
7	Vector Hash Aggregate	15.430,1.4793	1	1087848	1	15109	[1512KB, 1512KB]	1405	1
8	Costize Scan on dim_loc_bscrnc	11.071,1.229	1	1087848	1	15109	[1512KB, 1512KB]	1405	1
9	Vector Hash Join (1,2)	12461.130,2753.418	1	14334456	1	1287625	[2119KB, 2119KB]	1405	1
10	Costize Scan on sdr_web_bscrnc_1day sdr	12451.201,2751.045	1	14334456	1	1287625	[2119KB, 2119KB]	1405	1
11	Costize Scan on dim_rat_mapping rat	1.070,0.111	1	288	4	1577KB, 577KB	1	145	1.005024.25
(11 rows)									

Optimization Analysis

- As shown in Figure 6-12, the sequential scan phase is time consuming.
- The JOIN performance is poor because a large number of null values exist in the JOIN column **BSCRNC_ID** of the **PS.SDR_WEB_BSCRNC_1DAY** table.

Therefore, you are advised to manually add **NOT NULL** for **JOIN** columns in the statement, as shown below:

```
SELECT
*
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
INNER JOIN (SELECT
```

```
BSCRNC_ID,  
BSCRNC_NAME,  
ACCESS_TYPE,  
ACCESS_TYPE_ID  
FROM  
nethouse.DIM_LOC_BSCRNC  
GROUP BY  
BSCRNC_ID,  
BSCRNC_NAME,  
ACCESS_TYPE,  
ACCESS_TYPE_ID) DIM  
ON SDR.BSCRNC_ID = DIM.BSCRNC_ID  
AND DIM.ACCESS_TYPE_ID IN (0,1,2)  
INNER JOIN nethouse.DIM_RAT_MAPPING RAT  
ON (RAT.RAT = SDR.RAT)  
WHERE  
( ( STARTTIME >= 1461340800  
AND STARTTIME < 1461427200 )  
AND RAT.ACCESS_TYPE_ID IN (0,1,2)  
and SDR.BSCRNC_ID is not null  
GROUP BY  
STTIME ) ) A;
```

Figure 6-13 shows the execution plan.

Figure 6-13 Adding NOT NULL for JOIN columns (2)

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Row Adapter	[873,795]	1	72	72KB				140 121433605.45
2	-> Vector Streaming (type: GATHER)	[873,795]	1	72	44KB				140 121433605.45
3	-> Vector Hash Aggregate	[685,940,744,654]	1	1	[3004KB, 3004KB]	1MB	[75, 78]		85 1486577.84
4	-> Vector Streaming (type: REDISTRIBUTE)	[685,910,744,565]	72	1	[2424KB, 2424KB]	1MB			85 1486577.89
5	-> Vector Hash Aggregate	[590,319,710,912]	72	1	[5015KB, 5015KB]	1MB	[75, 78]		85 1486577.84
6	-> Vector Hash Join ("10")	[561,468,661,631]	3665920	102205	[2745KB, 2745KB]	1MB			85 1486533.77
7	-> Vector Hash Join ("9")	[541,846,636,604]	3664400	44859	[2333KB, 2333KB]	1MB			60 1396757.26
8	-> CStore Scan on adr_web_bscrnc_lday adr	[541,494,628,605]	3664400	78503	[3359KB, 3359KB]	1MB			64 1595824.20
9	-> CStore Scan on dim_rat_mapping rat	[10,051,0,107]	289	4	[577KB, 577KB]	1MB	[16, 16]		4 150.09
10	-> Vector Subquery Scan on dim	[10,326,6,560]	1087848	15109	[402KB, 402KB]	1MB	[19, 19]		7 1726.04
11	-> Vector Hash Aggregate	[5,497,6,931]	1087848	15109	[2539KB, 2539KB]	1MB	[48, 48]		32 1574.95
12	-> CStore Scan on dim_loc_bscrnc	[1,087,1,424]	1087848	15109	[1412KB, 1412KB]	1MB			32 1272.77

6.11.4 Case: Pushing Down Sort Operations to DN

Symptom

In an execution plan, more than 95% of the execution time is spent on **window agg** performed on the CN. In this case, **sum** is performed for the two columns separately, and then another **sum** is performed for the separate sum results of the two columns. After this, **trunc** and **sort** are performed in sequence.

The table structure is as follows:

```
CREATE TABLE public.test(imsi int,L4_DW_THROUGHPUT int,L4_UL_THROUGHPUT int)  
with (orientation = column) DISTRIBUTE BY hash(imsi);
```

The query statements are as follows:

```
SELECT COUNT(1) over() AS DATACNT,  
IMSI AS IMSI_IMSI,  
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))), 0) AS  
DECIMAL(20)) AS TOTAL_VOLUME_KPIID  
FROM public.test AS test  
GROUP BY IMSI  
order by TOTAL_VOLUME_KPIID DESC;
```

The execution plan is as follows: (Note: You are advised not to use vectorized execution operators.)

```
Row Adapter (cost=10.70..10.70 rows=10 width=12)  
-> Vector Sort (cost=10.68..10.70 rows=10 width=12)  
Sort Key: ((trunc(((sum(l4_ul_throughput)) + (sum(l4_dw_throughput))))::numeric,  
0))::numeric(20,0))
```

```
-> Vector WindowAgg (cost=10.09..10.51 rows=10 width=12)
-> Vector Streaming (type: GATHER) (cost=242.04..246.84 rows=240 width=12)
    Node/s: All datanodes
-> Vector Hash Aggregate (cost=10.09..10.29 rows=10 width=12)
    Group By Key: imsi
-> CStore Scan on test (cost=0.00..10.01 rows=10 width=12)
```

As we can see, both **window agg** and **sort** are performed on the CN, which is time consuming.

Optimization Analysis

Modify the statement to a subquery statement, as shown below:

```
SELECT COUNT(1) over() AS DATACNT, IMSI_IMSI, TOTAL_VOLOME_KPIID
FROM (SELECT IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))),
0) AS DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
ORDER BY TOTAL_VOLOME_KPIID DESC);
```

Perform **sum** on the **trunc** results of the two columns, take it as a subquery, and then perform **window agg** for the subquery to push down the sorting operation to DNs, as shown below:

Note: You are advised not to use vectorized executor operators.

```
Row Adapter (cost=10.70..10.70 rows=10 width=24)
-> Vector WindowAgg (cost=10.45..10.70 rows=10 width=24)
-> Vector Streaming (type: GATHER) (cost=250.83..253.83 rows=240 width=24)
    Node/s: All datanodes
-> Vector Sort (cost=10.45..10.48 rows=10 width=12)
    Sort Key: ((trunc(((sum(test.l4_ul_throughput) + sum(test.l4_dw_throughput))))::numeric,
0))::numeric(20,0))
-> Vector Hash Aggregate (cost=10.09..10.29 rows=10 width=12)
    Group By Key: test.imsi
-> CStore Scan on test (cost=0.00..10.01 rows=10 width=12)
```

The optimized SQL statement greatly improves the performance by reducing the execution time from 120s to 7s.

6.11.5 Case: Setting cost_param and Optimizing Query Performance

Symptom 1

If **bit0** of **cost_param** is set to **1** (**set cost_param=1**), an improved mechanism is used for estimating the selectivity of non-equi-joins. This method is more accurate for estimating the selectivity of joins between two identical tables. The following example describes the optimization scenario when **bit0** of **cost_param** is set to **1**. At present, if **cost_param & 1** is set to a value other than 0, the path is not used. That is, an optimized formula is selected for calculation.

Note: The selectivity indicates the percentage for which the number of rows meeting the join conditions account of the **JOIN** results when the **JOIN** relationship is established between two tables.

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;
```

The following figure shows the execution plan. (When **verbose** is used, **distinct** is added for column selection which is controlled by **cost off/on**. The hash join rows show the estimated number of **distinct** values and the other rows do not.)

id	operation	E-rows	E-distinct	E-width	E-costs
1	-> Row Adapter	1		8	39.36
2	-> Vector Sort	1		8	39.36
3	-> Vector Aggregate	1		8	39.34
4	-> Vector Streaming (type: GATHER)	2		8	39.34
5	-> Vector Aggregate	2		8	39.25
6	-> Vector Hash Anti Join (7, 10)	2	4, 5	0	39.24
7	-> Vector Hash Join (8,9)	2	200, 1	16	26.12
8	-> CStore Scan on public.lineitem 11	7		16	13.05
9	-> CStore Scan on public.orders	1		8	13.05
10	-> CStore Scan on public.lineitem 13	7		16	13.05

(10 rows)

Optimization Analysis 1

These queries are from Anti Join connected in the **lineitem** table. When **bit0** of **cost_param** is set to **1**, the estimated number of Anti Join rows greatly differ from that of the actual number of rows so that the query performance deteriorates. You can estimate the number of Anti Join rows more accurately by setting **bit0** of **cost_param** to **1** to improve the query performance. The optimized execution plan is as follows:

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	1		0	9104892.37 9
2	-> Vector Sort	1		0	9104892.37 9
3	-> Vector Aggregate	1		0	9104892.35 8
4	-> Vector Streaming (type: GATHER)	48		0	9104892.35 8
5	-> Vector Aggregate	48	1MB	0	9104890.82 5
6	-> Vector Hash Join (7,12)	2526630903	929MB	0	8973295.45 4
7	-> Vector Hash Anti Join (8, 10)	1999996587	3178MB	8	7198231.14
8	-> Vector Partition Iterator	1999996587	1MB	16	3000158.25
9	-> Partitioned CStore Scan on public.lineitem 11	1999996587	1MB	16	3000158.25 1
10	-> Vector Partition Iterator	1999996587	1MB	16	3000158.25
11	-> Partitioned CStore Scan on public.lineitem 13	1999996587	1MB	16	3000158.25
12	-> Vector Partition Iterator	730839014	1MB	8	589611.00
13	-> Partitioned CStore Scan on public.orders	730839014	1MB	8	589611.00

(13 rows)

Symptom 2

If **bit1** of **cost_param** is set to **1** (set **cost_param=2**), the selectivity is estimated based on multiple filter criteria. The lowest selectivity among all filter criteria, but not the product of the selectivities for two tables under a specific filter criterion, is used as the total selectivity. This method is more accurate when a close correlation exists between the columns to be filtered. The following example describes the optimization scenario when **bit1** of **cost_param** is set to **1**.

The table structure is as follows:

```
CREATE TABLE NATION
(
  N_NATIONKEY INT NOT NULL
, N_NAME      CHAR(25) NOT NULL
, N_REGIONKEY INT NOT NULL
, N_COMMENT   VARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
  S_SUPPKEY   BIGINT NOT NULL
, S_NAME      CHAR(25) NOT NULL
, S_ADDRESS   VARCHAR(40) NOT NULL
```

```

, S_NATIONKEY INT NOT NULL
, S_PHONE CHAR(15) NOT NULL
, S_ACCTBAL DECIMAL(15,2) NOT NULL
, S_COMMENT VARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
  PS_PARTKEY BIGINT NOT NULL
, PS_SUPPKEY BIGINT NOT NULL
, PS_AVAILQTY BIGINT NOT NULL
, PS_SUPPLYCOST DECIMAL(15,2) NOT NULL
, PS_COMMENT VARCHAR(199) NOT NULL
) distribute by hash(PS_PARTKEY);

```

The query statements are as follows:

```

set cost_param=2;
explain verbose select
nation,
sum(amount) as sum_profit
from
(
select
n_name as nation,
l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
from
supplier,
lineitem,
partsupp,
nation
where
s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and ps_partkey = l_partkey
and s_nationkey = n_nationkey
) as profit
group by nation
order by nation;

```

When **bit1** of **cost_param** is set to **0**, the execution plan is shown as follows:

id	operation	E-rows	E-distinct	E-width	E-costs
1	-> Sort	1		208	61.52
2	-> HashAggregate	1		208	61.51
3	-> Streaming (type: GATHER)	2		208	61.51
4	-> HashAggregate	2		208	61.36
5	-> Hash Join (6,7)	2	20, 15	176	61.33
6	-> Seq Scan on public.nation	40		108	20.20
7	-> Hash	2		76	41.04
8	-> Hash Join (9,16)	2	10, 13	76	41.04
9	-> Streaming(type: REDISTRIBUTE)	2		88	27.73
10	-> Hash Join (11,14)	2	10, 13	88	27.62
11	-> Streaming(type: REDISTRIBUTE)	20		70	14.19
12	-> Row Adapter	21		70	13.01
13	-> CStore Scan on public.lineitem	20		70	13.01
14	-> Hash	21		34	13.13
15	-> Seq Scan on public.partsupp	20		34	13.13
16	-> Hash	21		12	13.13
17	-> Seq Scan on public.supplier	20		12	13.13
(17 rows)					

Optimization Analysis 2

In the preceding queries, the hash join criteria of the **supplier**, **lineitem**, and **partsupp** tables are setting **lineitem.l_suppkey** to **supplier.s_suppkey** and **lineitem.l_partkey** to **partsupp.ps_partkey**. Two filter criteria exist in the hash join conditions. **lineitem.l_suppkey** in the first filter criterion and **lineitem.l_partkey** in the second filter criterion are two columns with strong relationship of the **lineitem** table. In this situation, when you estimate the rate of

the hash join conditions, if **bit1** of **cost_param** is set to **0**, the selectivity is estimated based on multiple filter criteria. The lowest selectivity among all filter criteria, but not the product of the selectivities for two tables under a specific filter criterion, is used as the total selectivity. Therefore, you need to set **bit1** of **cost_param** to **2** and select the lowest selectivity as the total selectivity to optimize the query performance. The optimized query plan is shown in the following figure.

id	operation	E-rows	E-distinct	E-width	E-costs
1	-> Sort	10		208	64.42
2	-> HashAggregate	10		208	64.23
3	-> Streaming (type: GATHER)	20		208	64.23
4	-> HashAggregate	20		208	62.71
5	-> Hash Join (6,7)	20 20, 10		176	62.46
6	-> Seq Scan on public.nation	40		108	20.20
7	-> Hash	20		76	41.97
8	-> Hash Join (9,16)	20 10, 13		76	41.97
9	-> Streaming(type: REDISTRIBUTE)	20		82	28.54
10	-> Hash Join (11,14)	20 10, 13		82	27.63
11	-> Streaming(type: REDISTRIBUTE)	20		70	14.19
12	-> Row Adapter	21		70	13.01
13	-> CStore Scan on public.lineitem	20		70	13.01
14	-> Hash	21		12	13.13
15	-> Seq Scan on public.supplier	20		12	13.13
16	-> Hash	21		34	13.13
17	-> Seq Scan on public.partsupp	20		34	13.13

(17 rows)

6.11.6 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]	5586	670934	[2091KB, 2093KB]	1MB		45	102533305.05
7	-> Hash Join (8,12)	[2657.406,93339.924]	5586	670934	[20KB, 20KB]	1MB		45	102532655.39
8	-> Seq Scan on s_riskrate_setting a	[682385,7340389]		77252027	78594218	[611KB, 903KB]	1MB	56	275264.71
9	-> Hash	[2241.418,2713.381]	8536241	8536241	[1031KB, 97803KB]	10MB	[48,48]	46	50870.88
10	-> Streaming (type: REDISTRIBUTE)	[210.226,2617.195]	8536241	8536241	[2091KB, 2093KB]	1MB		46	50870.88
11	-> Seq Scan on temp_calc_emprate0101 b	[86.790,141.293]	8536241	8536241	[16KB, 16KB]	1MB		46	11564.79

(11 rows)

According to the execution information, Hash Join becomes the performance bottleneck of the whole plan. Based on the execution time of Hash Join **[2657.406,93339.924]** (for details about the value, see [Description](#)), it can be seen that severe skew occurs on different DN's 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
Datatype:
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.

6.11.7 Case: Adjusting Partial Clustering Keys

Symptom

Information on the **EXPLAIN PERFORMANCE** at a site is as follows: As shown in the red boxes, two performance bottlenecks are scan operations in a table.

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	→ Row Adapter	14377.760	1	96	158KB				112 1097180.70
2	→ Vector Streaming (type: GATHER)	14377.760	1	96	1543KB				112 1097180.70
3	→ Vector Sort	14348.464,14348.464	1	96	1849KB, 3439KB				112 1097177.61
4	→ Vector Hash Aggregate	14348.289,14348.289	1	96	13068KB, 3068KB			[0, 96]	132 1097177.55
5	→ Vector Append	14347.522,14347.522	1	96	113B, 12B				132 1097177.42
6	→ Vector Subquery Scan on "SELECT" 1*	110112.875,10112.875	0	32	198KB, 98KB				136 649136.85
7	→ Vector Sort Aggregate	10112.871,10112.871	0	32	11525KB, 1525KB				136 649136.84
8	→ Vector Nest Loop Semi Join (5, 13)	110112.883,10112.883	0	2	1480KB, 480KB				136 649136.73
9	→ Vector Append	110112.880,10112.880	0	2	128B, 28B				136 649085.87
10	→ Partitioned Dfs Scan on pd_data_ap_app_acct_sec_trade_day	110112.872,10112.872	0	1	12844KB, 2944KB				87 649081.68
11	→ Vector Adapter	10.008,0.008	0	1	1114KB, 114KB				212 33.92
12	→ Seq Scan on csaosex_pg_delta_0278763770 app_acct_sec_trade_day	10.002,0.002	0	1	137KB, 37KB				212 33.92
13	→ Vector Materialize	10,0	0	2	10, 0				11 81.13
14	→ Vector Append	10,0	0	2	10, 0				11 81.12
15	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	10,0	0	1	10, 0				11 48.91
16	→ Vector Adapter	10,0	0	1	10, 0				11 2.21
17	→ Seq Scan on csaosex_pg_delta_4036128044 acct_acct_opt_his	10,0	0	1	10, 0				11 2.21
18	→ Vector Subquery Scan on "SELECT" 2*	119.036,19.036	0	32	198KB, 98KB				132 18809.79
19	→ Vector Sort Aggregate	119.036,19.036	0	32	11520KB, 1520KB				136 18809.78
20	→ Vector Nest Loop Semi Join (21, 28)	119.897,19.897	0	2	1480KB, 480KB				136 18809.67
21	→ Vector Append	119.894,19.894	0	2	128B, 28B				136 18782.51
22	→ Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_03	118.886,18.886	0	1	11439KB, 1439KB				97 18718.89
23	→ Vector Adapter	10.004,0.004	0	1	1114KB, 114KB				212 33.92
24	→ Seq Scan on csaosex_pg_delta_638785702 app_acct_sec_trade_day_03	10.002,0.002	0	1	137KB, 37KB				212 33.92
25	→ Vector Materialize	10,0	0	2	10, 0				11 81.13
26	→ Vector Append	10,0	0	2	10, 0				11 81.12
27	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	10,0	0	1	10, 0				11 48.91
28	→ Vector Adapter	10,0	0	1	10, 0				11 2.21
29	→ Seq Scan on csaosex_pg_delta_4036128044 acct_acct_opt_his	10,0	0	1	10, 0				11 2.21
30	→ Vector Subquery Scan on "SELECT" 3*	14216.008,4216.008	1	32	198KB, 98KB				132 432236.79
31	→ Vector Sort Aggregate	14216.004,4216.004	1	32	17913KB, 17913KB				136 432236.78
32	→ Vector Nest Loop Semi Join (33, 37)	14215.608,4215.608	1	2	1480KB, 480KB				136 432236.67
33	→ Vector Append	14212.885,4212.885	1	2	128B, 28B				136 432185.50
34	→ Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_02	14212.845,4212.845	1	1	12400KB, 2600KB				96 432181.58
35	→ Vector Adapter	10.006,0.006	0	1	1114KB, 114KB				212 33.92
36	→ Seq Scan on csaosex_pg_delta_1204972051 app_acct_sec_trade_day_02	10.002,0.002	0	1	137KB, 37KB				212 33.92
37	→ Vector Materialize	12.759,2.759	0	1	1242KB, 242KB			[0, 17]	11 81.19
38	→ Vector Append	12.616,2.616	1	2	1848KB, 848KB				11 81.12
39	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	12.614,2.614	1	1	118KB, 18KB				11 48.91
40	→ Vector Adapter	10,0	0	1	10, 0				11 2.21
41	→ Seq Scan on csaosex_pg_delta_4036128044 acct_acct_opt_his	10,0	0	1	10, 0				11 2.21

Optimization Analysis

Based on further analysis, the filter condition **acct_id = 'A012709548'::bpchar** exists in **Scan** of two tables.

```
--Partitioned Dfs Scan on pd_data_ap_app_acct_sec_trade_day
Filter: ((pd_data_ap_app_acct_sec_trade_day.acct_id ~ '584'::text) AND (((CASE WHEN (pd_data_ap_app_acct_sec_trade_day.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((pd_data_ap_app_acct_sec_trade_day.buy_vol ~ pd_data_ap_app_acct_sec_trade_day.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (pd_data_ap_app_acct_sec_trade_day.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_0278763770 app_acct_sec_trade_day
Filter: ((csaosex_pg_delta_0278763770.app_acct_sec_trade_day.acct_id ~ '584'::text) AND (csaosex_pg_delta_0278763770.app_acct_sec_trade_day.acct_id ~ 'A012709548'::bpchar) AND (((CASE WHEN (csaosex_pg_delta_0278763770.app_acct_sec_trade_day.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((pd_data_ap_app_acct_sec_trade_day.buy_vol ~ pd_data_ap_app_acct_sec_trade_day.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (pd_data_ap_app_acct_sec_trade_day.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_4036128044 acct_acct_opt_his
Filter: (csaosex_pg_delta_4036128044.acct_acct_opt_his.acct_id ~ 'A012709548'::bpchar)

--Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_03
Filter: ((hd_data_ap_app_acct_sec_trade_day_03.sec_code ~ '584'::text) AND (((CASE WHEN (hd_data_ap_app_acct_sec_trade_day_03.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((hd_data_ap_app_acct_sec_trade_day_03.buy_vol ~ hd_data_ap_app_acct_sec_trade_day_03.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (hd_data_ap_app_acct_sec_trade_day_03.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_638785702 app_acct_sec_trade_day_03
Filter: ((csaosex_pg_delta_638785702.app_acct_sec_trade_day_03.acct_id ~ '584'::text) AND (csaosex_pg_delta_638785702.app_acct_sec_trade_day_03.acct_id ~ 'A012709548'::bpchar) AND (((CASE WHEN (csaosex_pg_delta_638785702.app_acct_sec_trade_day_03.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((hd_data_ap_app_acct_sec_trade_day_03.buy_vol ~ hd_data_ap_app_acct_sec_trade_day_03.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (hd_data_ap_app_acct_sec_trade_day_03.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_1204972051 app_acct_sec_trade_day_02
Filter: ((hd_data_ap_app_acct_sec_trade_day_02.sec_code ~ '584'::text) AND (((CASE WHEN (hd_data_ap_app_acct_sec_trade_day_02.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((hd_data_ap_app_acct_sec_trade_day_02.buy_vol ~ hd_data_ap_app_acct_sec_trade_day_02.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (hd_data_ap_app_acct_sec_trade_day_02.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_1204972051 app_acct_sec_trade_day_02
Filter: ((csaosex_pg_delta_1204972051.app_acct_sec_trade_day_02.acct_id ~ '584'::text) AND (csaosex_pg_delta_1204972051.app_acct_sec_trade_day_02.acct_id ~ 'A012709548'::bpchar) AND (((CASE WHEN (csaosex_pg_delta_1204972051.app_acct_sec_trade_day_02.sec_code ~ ANY ('{'2018,2024,2048'}::text[])) THEN ((hd_data_ap_app_acct_sec_trade_day_02.buy_vol ~ hd_data_ap_app_acct_sec_trade_day_02.acct_id ~ 'A012709548'::bpchar)
Randomize Predicate Filter: (pd_data_act_acct_opt_his.acct_id ~ 'A012709548'::bpchar)

--Seq Scan on csaosex_pg_delta_4036128044 acct_acct_opt_his
Filter: (csaosex_pg_delta_4036128044.acct_acct_opt_his.acct_id ~ 'A012709548'::bpchar)
```

Try to add the partial clustering key in the **acct_id** column of the two tables, and run the **VACUUM FULL** statement to make the local clustering take effect. The table performance is improved.

6.11.8 Case: Modifying a Partitioned Table

Symptom

In the following simple SQL statements, the performance bottlenecks exist in the scan operation of the **dwjck** table.

```
openGauss=# explain performance select zqdh, count(1) from dwjck where cjrj = '2015-05-02 00:00:00' group by zqdh;
```

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	1599.794	58	12	19KB				7 771106.83
2	-> Vector Streaming (type: GATHER)	1599.781	58	12	210KB				7 771106.83
3	-> Vector Hash Aggregate	[1445.092,1446.332]	58	2	[2315KB, 2315KB]	16MB	[16,16]		7 128517.80
4	-> Vector Streaming(type: REDISTRIBUTE)	[1444.996,1446.259]	340	12	[247KB, 247KB]	1MB			7 128518.09
5	-> Vector Hash Aggregate	[573.150,1261.354]	340	12	[2297KB, 2297KB]	16MB	[16,16]		7 128517.80
6	-> CStore Scan on public.dwjck	[330.178,1021.695]	10000000	1623137	[786KB, 786KB]	1MB			7 120402.00

Optimization Analysis

Obviously, there are date features in the **cjrj** column of table data in the service layer, and this meets the features of a partitioned table. Replan the table definition of the **dwjck** table. Set the **cjrj** column as a partition key, and day as an interval unit. Define the partitioned table **dwjck_part**. The modified result is as follows, and the performance is nearly doubled.

```
openGauss=# explain performance select zqdh, count(1) from dwjck_part where cjrj = '2015-05-02 00:00:00' group by zqdh;
```

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	977.457	58	14	19KB				7 773142.84
2	-> Vector Streaming (type: GATHER)	977.437	58	14	210KB				7 773142.84
3	-> Vector Hash Aggregate	[651.238,734.931]	58	2	[2316KB, 2316KB]	16MB	[16,16]		7 128857.14
4	-> Vector Streaming(type: REDISTRIBUTE)	[651.137,734.834]	340	14	[247KB, 247KB]	1MB			7 128857.47
5	-> Vector Hash Aggregate	[402.145,515.752]	340	14	[2297KB, 2297KB]	16MB	[16,16]		7 128857.14
6	-> Vector Partition Iterator	[142.630,275.890]	10000000	1691000	[3128YTE, 3128YTE]	1MB			7 120402.00
7	-> Partitioned CStore Scan on public.dwjck_part	[161.746,275.207]	10000000	1691000	[795KB, 795KB]	1MB			7 120402.00

6.11.9 Case: Adjusting the GUC Parameter best_agg_plan

Symptom

The **t1** table is defined as follows:

```
create table 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:

```
openGauss=# explain select a, count(1) from t1 group by a;
```

id	operation	E-rows	E-width	E-costs
1	-> Streaming (type: GATHER)	30	4	15.56
2	-> HashAggregate	30	4	14.31
3	-> Seq Scan on t1	30	4	14.14

(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.

```
openGauss=# set best_agg_plan to 1;
SET
openGauss=# 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)

openGauss=# set best_agg_plan to 2;
SET
openGauss=# 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)

openGauss=# set best_agg_plan to 3;
SET
openGauss=# 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.

6.11.10 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**.

6.11.11 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.

6.11.12 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 t2
WHERE 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 join operation. The execution plan is as follows:

```
Streaming (type: GATHER) (cost=1641429284.14..1641429523.98 rows=3840 width=49)
Node/s: All datanodes
-> Insert on channel calc_empfyc_c1_result_age_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(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
Spawn 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=449523360 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
Spawn 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:

```
select
ls_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 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	E-memory	A-width
1	-> Streaming (type: GATHER)	[6737,281]	0	192	[292KB]	1	
2	-> Insert on channel.calc_emptyc_cl_result_age_tmp	[4665,024,4990,666]	0	192	[1109KB, 1109KB]	1MB	
3	-> HashAggregate	[4664,996,4990,641]	0	192	[12KB, 12KB]	16MB	
4	-> Streaming (type: REDISTRIBUTE)	[4664,991,4990,637]	0	3392	[2090KB, 2090KB]	1MB	
5	-> HashAggregate	[3416,939,4958,348]	0	3392	[14KB, 14KB]	16MB	
6	-> Append	[3416,936,4958,340]	0	4011	[1KB, 1KB]	1MB	
7	-> Hash Join (0,9)	[2011,226,3080,697]	0	3947	[6KB, 6KB]	1MB	
8	-> Seq Scan on channel.p10_md_tmp_t2 t2	[803,785,1238,984]	443525717	443523360	[12KB, 12KB]	1MB	
9	-> Hash	[4,357,328,979]	252608	252608	[482KB, 482KB]	16MB	[35,39]
10	-> Streaming (type: BROADCAST)	[2,345,326,320]	252608	252608	[2090KB, 2090KB]	1MB	
11	-> Seq Scan on channel.calc_emptyc_cl_result_tmp_t1 t1	[0,011,0,030]	3947	3947	[11KB, 11KB]	1MB	
12	-> Hash Join (13,14)	[1276,238,2066,110]	0	44	[5KB, 5KB]	1MB	
13	-> Seq Scan on channel.p10_md_tmp_t2 t2	[777,552,1388,499]	443525717	443523360	[12KB, 12KB]	1MB	
14	-> Hash	[2,812,4,217]	252608	252608	[482KB, 482KB]	16MB	[25,27]
15	-> Streaming (type: BROADCAST)	[1,276,1,868]	252608	252608	[2090KB, 2090KB]	1MB	
16	-> Seq Scan on channel.calc_emptyc_cl_result_tmp_t1 t1	[0,010,0,033]	3947	3947	[11KB, 11KB]	1MB	
(16 rows)							

Before the optimization, no result is returned for more than 1 hour. After the optimization, the result is returned within 7s.

6.11.13 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 DNs 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 DNs for the distributed execution, and the performance is poor.

The following is an example:

```
yshen=# set rewrite_rule='none';
SET
yshen=# 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

```
-----
Hash Join
Output: two_sum(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
(13 rows)
```

The **two_sum()** 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 two_sum 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.

```
yshen=# set rewrite_rule='partialpush';
SET
yshen=# 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:

```
yshen=# set rewrite_rule='none';
SET
yshen=# 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.

```
yshen=# set rewrite_rule='intargetlist';
SET
yshen=# 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;
```

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.

```
yshen=# set rewrite_rule='uniquecheck';
SET
yshen=# 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:

```
yshen=# 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)
```

```
yshen=# 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.

```
yshen=# 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)
```

```
yshen=# set rewrite_rule='uniquecheck';
```

```
SET
```

```
yshen=# 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:

```
openGauss=# set enable_fast_query_shipping=off; -- Disable FQS optimization.
SET
openGauss=# show rewrite_rule;
rewrite_rule
-----
magicset
(1 row)

openGauss=# 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)

openGauss=# set rewrite_rule='predpushnormal';
SET
openGauss=# 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.
```

```

openGauss=# set rewrite_rule='predpushforce';
SET

openGauss=# explain (costs off) select /*+predpush(t1 st2)*/ * 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.

openGauss=# set rewrite_rule = 'predpush';
SET
openGauss=# 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)

```


7 SQL Reference

7.1 GaussDB 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. SQL 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 (SQL2)
- 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

SQL statements contain 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, or column aliases, regardless of whether they are enclosed with double quotation marks. In addition, they cannot be used as function names without 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` with 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
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
BARRIER	Non-reserved	N/A	N/A
BEFORE	Non-reserved	Reserved	N/A
BEGIN	Non-reserved	Reserved	Reserved
BEGIN_NON_ANOYBLOCK	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BETWEEN	Non-reserved (excluding functions and types)	Non-reserved	Reserved
BIGINT	Non-reserved (excluding functions and types)	N/A	N/A
BINARY	Reserved (functions and types allowed)	Reserved	N/A
BINARY_DOUBLE	Non-reserved (excluding functions and types)	N/A	N/A
BINARY_INTEGER	Non-reserved (excluding functions and types)	N/A	N/A
BIT	Non-reserved (excluding functions and 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
BLOCKCHAIN	Non-reserved	N/A	N/A
BODY	Non-reserved	N/A	N/A
BOOLEAN	Non-reserved (excluding functions and types)	Reserved	N/A
BOTH	Reserved	Reserved	Reserved
BREADTH	N/A	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BUCKETCNT	Non-reserved (excluding functions and types)	N/A	N/A
BUCKETS	Reserved	N/A	N/A
BY	Non-reserved	Reserved	Reserved
BYTEAWITHOUTORDER	Non-reserved (excluding functions and types)	N/A	N/A
BYTEAWITHOUTORDER- WITHEQUAL	Non-reserved (excluding functions and 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
CHAR	Non-reserved (excluding functions and types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
CHAR_LENGTH	N/A	Non-reserved	Reserved
CHARACTER	Non-reserved (excluding functions and 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
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 (excluding functions and types)	Non-reserved	Reserved
COBOL	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
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
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
COMPLETE	Non-reserved	N/A	N/A
COMPLETION	N/A	Reserved	N/A
CONCURRENTLY	Reserved (functions and types allowed)	N/A	N/A
CONDITION	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
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
CONTAINS	N/A	Non-reserved	N/A
CONTENT	Non-reserved	N/A	N/A
CONTINUE	Non-reserved	Reserved	Reserved
CONVIEW	Non-reserved	N/A	N/A
CONVERSION	Non-reserved	N/A	N/A
CONVERT	N/A	Non-reserved	Reserved
COORDINATOR	Non-reserved	N/A	N/A
COORDINATORS	Non-reserved	N/A	N/A
COPY	Non-reserved	N/A	N/A
CORRESPONDING	N/A	Reserved	Reserved
COST	Non-reserved	N/A	N/A
COUNT	N/A	Non-reserved	Reserved
CREATE	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
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
DATATYPE_CL	Non-reserved	N/A	N/A
DATE	Non-reserved (excluding functions and types)	Reserved	Reserved
DATE_FORMAT	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
DATETIME_INTERVAL_CODE	N/A	Non-reserved	Non-reserved
DATETIME_INTERVAL_PRECISION	N/A	Non-reserved	Non-reserved
DAY	Non-reserved	Reserved	Reserved
DBCOMPATIBILITY	Non-reserved	N/A	N/A
DEALLOCATE	Non-reserved	Reserved	Reserved
DEC	Non-reserved (excluding functions and types)	Reserved	Reserved
DECIMAL	Non-reserved (excluding functions and types)	Reserved	Reserved
DECLARE	Non-reserved	Reserved	Reserved
DECODE	Non-reserved (excluding functions and 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
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

Keyword	GaussDB	SQL:1999	SQL-92
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
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
DROP	Non-reserved	Reserved	Reserved
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

Keyword	GaussDB	SQL:1999	SQL-92
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
ENFORCED	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
ERRORS	Non-reserved	N/A	N/A
ESCAPE	Non-reserved	Reserved	Reserved
ESCAPING	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

Keyword	GaussDB	SQL:1999	SQL-92
EXISTS	Non-reserved (excluding functions and types)	Non- reserved	Reserved
EXPIRED	Non-reserved	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 (excluding functions and types)	Non- reserved	Reserved
FALSE	N/A	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
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 (excluding functions and types)	Reserved	Reserved
FOLLOWING	Non-reserved	N/A	N/A
FOR	Reserved	Reserved	Reserved
FORCE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
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
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 (excluding functions and types)	N/A	N/A
GROUP	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
GROUPING	Non-reserved (excluding functions and 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
HOURL	Non-reserved	Reserved	Reserved
IDENTIFIED	Non-reserved	N/A	N/A
IDENTITY	Non-reserved	Reserved	Reserved
IF	Non-reserved	N/A	N/A
IGNORE	N/A	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
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

Keyword	GaussDB	SQL:1999	SQL-92
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
INITTRANS	Non-reserved	N/A	N/A
INLINE	Non-reserved	N/A	N/A
INNER	Reserved (functions and types allowed)	Reserved	Reserved
INOUT	Non-reserved (excluding functions and 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 (excluding functions and types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
INTEGER	Non-reserved (excluding functions and types)	Reserved	Reserved
INTERNAL	Non-reserved	N/A	N/A
INTERSECT	Reserved	Reserved	Reserved
INTERVAL	Non-reserved (excluding functions and 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
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

Keyword	GaussDB	SQL:1999	SQL-92
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 (excluding functions and 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
LIKE	Reserved (functions and types allowed)	Reserved	Reserved
LIMIT	Reserved	Reserved	N/A
LIST	Non-reserved	N/A	N/A
LISTEN	Non-reserved	N/A	N/A
LOAD	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

Keyword	GaussDB	SQL:1999	SQL-92
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
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

Keyword	GaussDB	SQL:1999	SQL-92
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
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 (excluding functions and types)	Reserved	Reserved
NATURAL	Reserved (functions and types allowed)	Reserved	Reserved
NCHAR	Non-reserved (excluding functions and 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
NOCYCLE	Reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NODE	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
NONE	Non-reserved (excluding functions and 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 (excluding functions and types)	Non- reserved	Reserved
NULLS	Non-reserved	N/A	N/A
NUMBER	Non-reserved (excluding functions and types)	Non- reserved	Non-reserved
NUMERIC	Non-reserved (excluding functions and types)	Reserved	Reserved
NUMSTR	Non-reserved	N/A	N/A
NVARCHAR2	Non-reserved (excluding functions and types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NVL	Non-reserved (excluding functions and 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	N/A	Reserved	N/A
OUT	Non-reserved (excluding functions and types)	Reserved	N/A
OUTER	Reserved (functions and types allowed)	Reserved	Reserved
OUTPUT	N/A	Reserved	Reserved
OVER	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
OVERLAPS	Reserved (functions and types allowed)	Non-reserved	Reserved
OVERLAY	Non-reserved (excluding functions and 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
PACKAGE	Non-reserved	N/A	N/A
PACKAGES	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

Keyword	GaussDB	SQL:1999	SQL-92
PASSWORD	Non-reserved	N/A	N/A
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
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
POOL	Non-reserved	N/A	N/A
POSITION	Non-reserved (excluding functions and types)	Non-reserved	Reserved
POSTFIX	N/A	Reserved	N/A
PRECEDING	Non-reserved	N/A	N/A
PRECISION	Non-reserved (excluding functions and 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

Keyword	GaussDB	SQL:1999	SQL-92
PRIORER	Reserved	N/A	N/A
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	N/A	Reserved	Reserved
PUBLICATION	Non-reserved	N/A	N/A
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 (excluding functions and types)	Reserved	Reserved
REASSIGN	Non-reserved	N/A	N/A
REBUILD	Non-reserved	N/A	N/A
RECHECK	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

Keyword	GaussDB	SQL:1999	SQL-92
REFERENCES	Reserved	Reserved	Reserved
REFERENCING	N/A	Reserved	N/A
REFRESH	Non-reserved	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
RESOURCE	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
REVOKE	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
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 (excluding functions and 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
SCHEMA	Non-reserved	Reserved	Reserved
SCHEMA_NAME	N/A	Non-reserved	Non-reserved
SCOPE	N/A	Reserved	N/A
SCROLL	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
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
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
SET	Non-reserved	Reserved	Reserved
SETOF	Non-reserved (excluding functions and 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
SHUTDOWN	Non-reserved	N/A	N/A
SIBLINGS	Non-reserved	N/A	N/A
SIMILAR	Reserved (functions and types allowed)	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SIMPLE	Non-reserved	Non-reserved	N/A
SIZE	Non-reserved	Reserved	Reserved
SKIP	Non-reserved	N/A	N/A
SLICE	Non-reserved	N/A	N/A
SMALLDATETIME	Non-reserved (excluding functions and types)	N/A	N/A
SMALLDATETIME_FORMAT	Non-reserved	N/A	N/A
SMALLINT	Non-reserved (excluding functions and 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
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
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

Keyword	GaussDB	SQL:1999	SQL-92
START	Non-reserved	Reserved	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
STREAM	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
SUBSCRIPTION	Non-reserved	N/A	N/A
SUBSTRING	Non-reserved (excluding functions and types)	Non-reserved	Reserved
SUM	N/A	Non-reserved	Reserved
SYMMETRIC	Reserved	Non-reserved	N/A
SYNONYM	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
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
TIME	Non-reserved (excluding functions and types)	Reserved	Reserved
TIME_FORMAT	Non-reserved	N/A	N/A
TIMECAPSULE	Reserved (functions and types allowed)	N/A	N/A
TIMESTAMP	Non-reserved (excluding functions and types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
TIMESTAMP_FORMAT	Non-reserved	N/A	N/A
TIMESTAMPDIFF	Non-reserved (excluding functions and types)	N/A	N/A
TIMEZONE_HOUR	N/A	Reserved	Reserved
TIMEZONE_MINUTE	N/A	Reserved	Reserved
TINYINT	Non-reserved (excluding functions and 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
TRANSLATION	N/A	Reserved	Reserved
TREAT	Non-reserved (excluding functions and 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

Keyword	GaussDB	SQL:1999	SQL-92
TRIGGER_SCHEMA	N/A	Non-reserved	N/A
TRIM	Non-reserved (excluding functions and types)	Non-reserved	Reserved
TRUE	N/A	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
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
UNTIL	Non-reserved	N/A	N/A
UNUSABLE	Non-reserved	N/A	N/A
UPDATE	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
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 (excluding functions and types)	Reserved	Reserved
VARCHAR	Non-reserved (excluding functions and types)	Reserved	Reserved
VARCHAR2	Non-reserved (excluding functions and 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

Keyword	GaussDB	SQL:1999	SQL-92
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
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
WRAPPER	Non-reserved	N/A	N/A
WRITE	Non-reserved	Reserved	Reserved
XML	Non-reserved	N/A	N/A
XMLATTRIBUTES	Non-reserved (excluding functions and types)	N/A	N/A
XMLCONCAT	Non-reserved (excluding functions and types)	N/A	N/A
XMLELEMENT	Non-reserved (excluding functions and types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
XML EXISTS	Non-reserved (excluding functions and types)	N/A	N/A
XML FOREST	Non-reserved (excluding functions and types)	N/A	N/A
XML PARSE	Non-reserved (excluding functions and types)	N/A	N/A
XML PI	Non-reserved (excluding functions and types)	N/A	N/A
XML ROOT	Non-reserved (excluding functions and types)	N/A	N/A
XML SERIALIZE	Non-reserved (excluding functions and 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

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 defines the data type. During storage, 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 [Mathematical Functions and Operators](#).

Table 7-2 Integer types

Name	Description	Storage Space	Range
TINYINT	Tiny integer, also called INT1	1 byte	0 to 255
SMALLINT	Small integer, also called INT2	2 bytes	-32,768 to +32,767
INTEGER	Typical choice for integers, also called INT4	4 bytes	-2,147,483,648 to +2,147,483,647
BINARY_INTEGER	INTEGER alias, which is an ORA-compatible database type	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.
openGauss=# CREATE TABLE int_type_t1
(
  IT_COL1 TINYINT
);

-- Insert data.
openGauss=# INSERT INTO int_type_t1 VALUES(10);

-- View data.
openGauss=# SELECT * FROM int_type_t1;
it_col1
-----
10
(1 row)
```

```
-- Drop the table.
openGauss=# DROP TABLE int_type_t1;
-- Create a table containing TINYINT, INTEGER, and BIGINT data.
openGauss=# CREATE TABLE int_type_t2
(
  a TINYINT,
  b TINYINT,
  c INTEGER,
  d BIGINT
);

-- Insert data.
openGauss=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

-- View data.
openGauss=# SELECT * FROM int_type_t2;
 a | b | c | d
-----+-----
100 | 10 | 1000 | 10000
(1 row)

-- Drop the table.
openGauss=# DROP TABLE int_type_t2;
```

NOTE

- Only 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.
openGauss=# CREATE TABLE decimal_type_t1
(
    DT_COL1 DECIMAL(10,4)
);

-- Insert data.
openGauss=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

-- Query data in the table.
openGauss=# SELECT * FROM decimal_type_t1;
 dt_col1
-----
123456.1223
(1 row)

-- Drop the table.
openGauss=# DROP TABLE decimal_type_t1;
-- Create a table.
openGauss=# CREATE TABLE numeric_type_t1
(
    NT_COL1 NUMERIC(10,4)
);

-- Insert data.
openGauss=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

-- Query data in the table.
openGauss=# SELECT * FROM numeric_type_t1;
 nt_col1
-----
123456.1235
(1 row)

-- Drop the table.
openGauss=# 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 only 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.
openGauss=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

-- Insert data.
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);

-- Insert data again.
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

-- Create a table.
openGauss=# CREATE TABLE serial_type_tab(b SERIAL);

-- Insert data.
openGauss=# INSERT INTO serial_type_tab VALUES(default);

-- Insert data again.
openGauss=# INSERT INTO serial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

-- Create a table.
openGauss=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

-- Insert data.
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);

-- Insert data again.
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

-- Drop the table.
openGauss=# DROP TABLE smallserial_type_tab;

openGauss=# DROP TABLE serial_type_tab;

openGauss=# 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 a **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, inexact	4 bytes.	–3.402E+38 to +3.402E+38, 6-bit decimal digits.
DOUBLE PRECISION , FLOAT8	Double precision floating points, inexact	8 bytes.	–1.79E+308 to +1.79E+308, 15-bit decimal digits.
FLOAT[(p)]	Floating points, inexact The value range of precision (p) is [1,53]. NOTE p is the precision, indicating the total number of binary bits.	4 bytes or 8 bytes.	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.
BINARY_DOUBLE	Alias of DOUBLE PRECISION.	8 bytes.	–1.79E+308 to +1.79E+308, 15-bit decimal digits.

Name	Description	Storage Space	Range
DEC[(p[,s])]	The value range of p is [1,1000], and the value range of s is [0,<i>p</i>]. If the scenario in the following "NOTE" is met and the precision and scale are not specified, the precision p is 10 and the scale s is 0 by default. This type maps to NUMERIC. For details about the application scenario, see NUMERIC.	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])]	The value range of p is [1,1000], and the value range of s is [0,<i>p</i>]. If the precision and scale are not specified, the precision p is 10 and the scale s is 0 by default. 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.	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. If the precision and scale are not specified, the value ranges from -2,147,483,648 to +2,147,483,647.

 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.

Example:

```
-- Create a table.
openGauss=# 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.
openGauss=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654,
123.123654);

-- View data.
openGauss=# 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.
openGauss=# DROP TABLE float_type_t2;
```

7.3.2 Monetary Types

The money 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:

```
openGauss=# 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:

```
openGauss=# 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, a pure number, not money); the currency units cancel each other out in the division.

7.3.3 Boolean Types

Table 7-7 Boolean type

Name	Description	Storage Space	Value
BOOLEAN	Boolean	1 byte	• true • false • null: unknown

Valid literal values for the "true" state are:

TRUE, **'t'**, **'true'**, **'y'**, **'yes'**, **'1'**, **'TRUE'**, **true**, **on**, and all non-zero integers.

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.
openGauss=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
)DISTRIBUTE BY HASH(BT_COL2);

-- Insert data.
openGauss=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

openGauss=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

-- View data.
openGauss=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
-----+-----
 t      | sic est
 f      | non est
(2 rows)

openGauss=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
-----+-----
 t      | sic est
(1 row)

-- Drop the table.
openGauss=# 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).
VARCHAR(n) CHARACTER VARYING(n)	Variable-length string. In PostgreSQL-compatible mode, n indicates the string length. In other compatibility modes, n indicates the byte length.	The maximum value of n is 10485760 (10 MB). If n is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first <i>n</i> columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).

Name	Description	Storage Space
VARCHAR2(n)	Variable-length string. It is an alias for VARCHAR(n) type, compatible with the Oracle Database. n indicates the string length.	The maximum value of n is 10485760 (10 MB). If n is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first <i>n</i> columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).
NVARCHAR2(n)	Variable-length string. n indicates the string length.	The maximum value of n is 10485760 (10 MB). If n is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first <i>n</i> columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).
CLOB	Big text object. It is compatible with the Oracle Database.	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.
TEXT	Variable-length string.	The maximum storage length is 1 GB – 85 bytes – Length of the first <i>n</i> columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).

 NOTE

1. In addition to the restriction on the size of each column, the total size of each tuple cannot exceed 1 GB minus 1 byte 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.

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 char type uses only 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

```
-- Create a table.
openGauss=# CREATE TABLE char_type_t1
(
    CT_COL1 CHARACTER(4)
)DISTRIBUTE BY HASH (CT_COL1);

-- Insert data.
openGauss=# INSERT INTO char_type_t1 VALUES ('ok');

-- Query data in the table.
openGauss=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
-----+-----
    ok   |          4
(1 row)

-- Delete the table.
openGauss=# DROP TABLE char_type_t1;
-- Create a table.
openGauss=# CREATE TABLE char_type_t2
(
    CT_COL1 VARCHAR(5)
)DISTRIBUTE BY HASH (CT_COL1);

-- Insert data.
openGauss=# INSERT INTO char_type_t2 VALUES ('ok');

openGauss=# INSERT INTO char_type_t2 VALUES ('good');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
openGauss=# 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.
openGauss=# INSERT INTO char_type_t2 VALUES ('too long::varchar(5));

-- Query data.
openGauss=# 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.
openGauss=# DROP TABLE char_type_t2;
```


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: - 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 .	The maximum size is 32 TB (35,184,372,088,832 bytes).
RAW	Variable-length hexadecimal string.	4 bytes plus the actual hexadecimal string. The maximum size is 1,073,733,621 bytes (that is, 1 GB minus 8,203 bytes).
BYTEA	Variable-length binary string.	4 bytes plus the actual binary string. The maximum size is 1,073,733,621 bytes (that is, 1 GB minus 8,203 bytes).

NOTE

In addition to the size limitation on each column, the total size of each tuple cannot exceed 1,073,733,621 bytes (that is, 1 GB minus 8,203 bytes).

Example:

```
-- Create a table.
openGauss=# CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) DISTRIBUTE BY REPLICATION;

-- Insert data.
openGauss=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\xDEADBEEF');

-- Query data in the table.
openGauss=# SELECT * FROM blob_type_t1;
```

```
bt_col1 | bt_col2 | bt_col3 | bt_col4
-----+-----+-----+-----
    10 |      | DEADBEEF | \xdeadbeef
(1 row)

-- Drop the table.
openGauss=# 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. 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 (without time zone). p indicates the precision after the decimal point. The value ranges from 0 to 6 . Minimum value: 00:00:00 . Maximum value: 24:00:00 .	8 bytes
TIME [(p)] [WITH TIME ZONE]	Time within one day (with time zone). p indicates the precision after the decimal point. The value ranges from 0 to 6 . Minimum value: 00:00:00+1559 . Maximum value: 24:00:00 .	12 bytes

Name	Description	Storage Space
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	Date and time (without a time zone). p indicates the precision after the decimal point. The value ranges from 0 to 6. Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). Maximum value: 294277-01-09AD 00:00:00.000000 (294277 A.D.).	8 bytes
TIMESTAMP[(p)] [WITH TIME ZONE]	Date and time (with time zone). TIMESTAMP is also called TIMESTAMPTZ. p indicates the precision after the decimal point. The value ranges from 0 to 6. Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). Maximum value: 294277-01-09AD 00:00:00.000000 (294277 A.D.). 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 often needs to modify the time zone file accordingly to ensure that the time is correct. 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. The capability of the same type of data in the database is different from that in an ORA-compatible database.	8 bytes
SMALLDATETIME	Date and time (without time zone). The precision level is minute. 31s to 59s are rounded into 1 minute. Minimum value: 4713-11-24BC 00:00:00.000000 (4713 B.C.). Maximum value: 294277-01-09AD 00:00:00.000000 (294277 A.D.).	8 bytes

Name	Description	Storage Space
INTERVAL DAY (l) TO SECOND (p)	Specifies the time interval (<i>X</i> days <i>X</i> hours <i>X</i> minutes <i>X</i> seconds). • l: indicates the precision of days. The value ranges from 0 to 6. The specific function is not implemented due to compatibility reasons. • 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)]	Time interval. • 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	Relative time interval. • The format is as follows: <i>X</i> years <i>X</i> mons <i>X</i> days <i>XX</i>:<i>XX</i>:<i>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
abstime	Date and time. • Format: <i>YYYY-MM-DD hh:mm:ss</i> +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.

Example:

```
-- Create a table.
openGauss=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
   coll
-----
2010-12-10
(1 row)

-- Drop the table.
openGauss=# DROP TABLE date_type_tab;

-- Create a table.
openGauss=# 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.
openGauss=# 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.
openGauss=# SELECT * FROM time_type_tab;
   da   |   dai   |   dfgh   |   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)

-- Drop the table.
openGauss=# DROP TABLE time_type_tab;

-- Create a table.
openGauss=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

-- Insert data.
openGauss=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

-- View data.
openGauss=# SELECT * FROM day_type_tab;
 a | b
---+---
 1 | 3 days
(1 row)

-- Drop the table.
openGauss=# DROP TABLE day_type_tab;

-- Create a table.
openGauss=# CREATE TABLE year_type_tab(a int, b interval year (6));

-- Insert data.
```

```
openGauss=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

-- View data.
openGauss=# SELECT * FROM year_type_tab;
 a |  b
---+-----
  1 | 2 years
(1 row)

-- Drop the table.
openGauss=# DROP TABLE year_type_tab;
```

Date Inputs

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

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. August 1 in DMY mode
1/18/1999	January 18 in MDY mode, rejected in other modes
01/02/03	- January 2, 2003 in MDY mode - February 1, 2003 in DMY mode - February 3, 2001 in YMD mode
1999-Jan-08	January 8 in any mode
Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in YMD mode, else error
08-Jan-99	January 8, except error in YMD mode
Jan-08-99	January 8, except error in YMD mode

Example	Description
19990108	ISO 8601: January 8, 1999 in any mode
990108	ISO 8601: January 8, 1999 in any mode
1999.008	Year and day of year
J2451187	Julian date
January 8, 99 BC	Year 99 BC

Example:

```
-- Create a table.
openGauss=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
      coll
-----
2010-12-10 00:00:00
(1 row)

-- View the date format.
openGauss=# SHOW datestyle;
 datestyle
-----
ISO, MDY
(1 row)

-- Set the date format.
openGauss=# SET datestyle='YMD';
SET

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
      coll
-----
2010-12-10 00:00:00
2010-12-11 00:00:00
(2 rows)

-- Drop the table.
openGauss=# 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:

```
openGauss=# SELECT time '04:05:06';
time
-----
04:05:06
(1 row)

openGauss=# SELECT time '04:05:06 PST';
time
-----
04:05:06
(1 row)
```



```
openGauss=# 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	Infinity, later than any other timestamp.
-infinity	timestamp	Infinity, earlier than any other timestamp.
now	date, time, and timestamp	Start time of the current transaction.
today	date and timestamp	Today at midnight.
tomorrow	date and timestamp	Tomorrow at midnight.
yesterday	date and timestamp	Yesterday at midnight.
allballs	time	00:00:00.00 UTC

Example:

```
-- Create a table.
openGauss=# CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, col4 time);

-- Insert data.
openGauss=# INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
openGauss=# INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NULL);

-- View data.
openGauss=# 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)

openGauss=# 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)
```

```
-----+-----+-----+-----+
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)

openGauss=# 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)

openGauss=# 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)

openGauss=# 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)

openGauss=# 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)

openGauss=# 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 the table.
openGauss=# 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 types

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.
openGauss=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

-- Insert data.
openGauss=# INSERT INTO reltime_type_tab VALUES ('90', '90');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
openGauss=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5 MONTHS 10 DAYS');
openGauss=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
openGauss=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

-- View data.
openGauss=# 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.
openGauss=# 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 x <i>n</i> bytes	Closed path (similar to polygon)	((x1,y1),...)
path	16 + 16 x <i>n</i> bytes	Open path	[(x1,y1),...]
polygon	40 + 16 x <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:

```
openGauss=# 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 syntax:

```
[ ( 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:

```
openGauss=# 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 syntax:

```
(( 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:

```
openGauss=# 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 syntax:

```
[ ( 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:

```
openGauss=# 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 syntax:

```
(( 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:

```
openGauss=# 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:

```
openGauss=# 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 **cidr** type (Classless Inter-Domain Routing) 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.

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
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:

```
openGauss=# CREATE TABLE cidr_test(id int, c cidr);
CREATE TABLE
openGauss=# INSERT INTO cidr_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (2, '192.168/24');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (3, '192.168/25');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (4, '192.168.1');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (5, '192.168');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (6, '10.1.2');
INSERT 0 1
```

```
openGauss=# INSERT INTO cidr_test VALUES (7, '10.1');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (8, '10');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (9, '2001:4f8:3:ba::/64');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (10, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (11, '::ffff:127.0.0.0/120');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (12, '::ffff:127.0.0.0/128');
INSERT 0 1
openGauss=# 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)

openGauss=# 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:

```
openGauss=# CREATE TABLE inet_test(id int, i inet);
CREATE TABLE
openGauss=# INSERT INTO inet_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (2, '192.168.100.128');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (3, '192.168.1.0/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (4, '192.168.1.0/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (5, '192.168.1.255/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (6, '192.168.1.255/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (7, '10.1.2.3/8');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (8, '11.1.2.3/16');
```



```
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (9, '12.1.2.3/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (10, '13.1.2.3/32');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (11, '2001:4f8:3:ba::/64');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (12, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (13, '::ffff:127.0.0.0/120');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (14, '::ffff:127.0.0.0/128');
INSERT 0 1
openGauss=# 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)

openGauss=# 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:

```
openGauss=# CREATE TABLE macaddr_test(id int, m macaddr);
CREATE TABLE
openGauss=# INSERT INTO macaddr_test VALUES (1, '08:00:2b:01:02:03');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (2, '08-00-2b-01-02-03');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (3, '08002b:010203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (4, '08002b-010203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (5, '0800.2b01.0203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (6, '08002b010203');
```

```
INSERT 0 1
openGauss=# 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)

openGauss=# 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. It is an error to attempt to store shorter or longer bit strings. 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)**, it will be truncated on the right if it is more than *n* bits.
- 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.
openGauss=# CREATE TABLE bit_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BIT(3),
    BT_COL3 BIT VARYING(5)
) DISTRIBUTE BY REPLICATION;

-- Insert data.
openGauss=# 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.
openGauss=# 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.
openGauss=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

-- View data.
openGauss=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
-----+-----+-----
    1 | 101    | 00
    2 | 100    | 101
(2 rows)
```

```
-- Drop the table.  
openGauss=# 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, 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.  
openGauss=# CREATE TABLE uuid_test(id int, test uuid) DISTRIBUTE BY HASH(test);  
  
-- Insert data in the example format.  
openGauss=# INSERT INTO uuid_test VALUES(1, 'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11');  
openGauss=# INSERT INTO uuid_test VALUES(2, '{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}');  
openGauss=# INSERT INTO uuid_test VALUES(3, 'a0eebc999c0b4ef8bb6d6bb9bd380a11');  
openGauss=# INSERT INTO uuid_test VALUES(4, 'a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11');  
  
-- View data and output the data in the standard format.  
openGauss=# 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 JSONB data is stored in its decomposed binary form, with semantic-irrelevant details and duplicate keys removed and key-values sorted. 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 its parsed 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.

```
openGauss=# SELECT 'null'::json; -- suc
json
-----
null
(1 row)

openGauss=# SELECT 'NULL'::jsonb; -- err
ERROR:  invalid input syntax for type json
```

Number (num-json): The value can be a positive or negative integer, decimal fraction, or 0. The scientific notation is supported.

```
openGauss=# SELECT '1'::json;
json
-----
1
(1 row)

openGauss=# SELECT '-1.5'::json;
json
-----
-1.5
(1 row)

openGauss=# SELECT '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
 jsonb | jsonb 
-----+-----
-0.000015 | -150
(1 row)

openGauss=# 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.

```
openGauss=# SELECT 'true'::json;
json
-----
true
(1 row)

openGauss=# SELECT 'false'::jsonb;
jsonb
-----
false
(1 row)
```

String (str-json): The value must be a string enclosed in double quotation marks ("").

```
openGauss=# SELECT '"a"'::json;
json
-----
"a"
(1 row)

openGauss=# SELECT '"abc"'::jsonb;
jsonb
-----
"abc"
(1 row)
```

Array (array-json): Arrays are enclosed in square brackets ([]). Elements in the array can be any valid JSON data, and are unnecessarily of the same type.

```
openGauss=# SELECT '[1, 2, "foo", null]'::json;
json
-----
[1, 2, "foo", null]
(1 row)

openGauss=# SELECT '[]'::json;
json
-----
[]
(1 row)

openGauss=# SELECT '[1, 2, "foo", null, [], {}]'::jsonb;
jsonb
-----
[1, 2, "foo", null, [], {}]
(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.

```
openGauss=# SELECT '{}'::json;
json
-----
{}
(1 row)

openGauss=# SELECT '{"a": 1, "b": {"a": 2, "b": null}}'::json;
json
-----
{"a": 1, "b": {"a": 2, "b": null}}
(1 row)

openGauss=# 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=""** 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 and MOTs 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:

```
openGauss=# 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:

```
openGauss=# select '{"a" : 1, "a" : 2} '::jsonb;
jsonb
-----
{"a": 2}
(1 row)
```
 - For object-json, key-values will be re-sorted. The collation 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:

```
openGauss=# 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 keys are compared first, and then the values are 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.

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

HyperLoglog (HLL) is an approximation algorithm for efficiently counting the number of distinct values in a data set. 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 data set, 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)

Item	Sorting Algorithm	Hash Algorithm	HLL
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 $O(n\log n)$ 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.
openGauss=# CREATE TABLE t1 (id integer, set hll);
openGauss=# \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.
openGauss=# CREATE TABLE t2 (id integer, set hll(12,4));
openGauss=# \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.
openGauss=# CREATE TABLE t3(id int, set hll(-1,-1,8,-1));
openGauss=# \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.
openGauss=# 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.
openGauss=# 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.
openGauss=# CREATE TABLE t1(id integer, set hll(14));

-- Insert an HLL object to a table. The insertion succeeds because parameter types are consistent.
openGauss=# INSERT INTO t1 VALUES (1, hll_empty(14,-1));
```

```
-- Insert an HLL object to a table. The insertion fails because parameter types are inconsistent.
openGauss=# INSERT INTO t1(id, set) VALUES (1, hll_empty(14,5));
ERROR: log2explicit does not match: source is 5 and dest is 10

-- Drop the table.
openGauss=# 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.
openGauss=# CREATE TABLE helloworld (id integer, set hll);

-- Insert an empty HLL to the table.
openGauss=# INSERT INTO helloworld(id, set) VALUES (1, hll_empty());

-- Add a hashed integer to the HLL.
openGauss=# UPDATE helloworld SET set = hll_add(set, hll_hash_integer(12345)) WHERE id = 1;

-- Add a hashed string to the HLL.
openGauss=# UPDATE helloworld SET set = hll_add(set, hll_hash_text('hello world')) WHERE id = 1;

-- Obtain the number of distinct values of the HLL.
openGauss=# SELECT hll_cardinality(set) FROM helloworld WHERE id = 1;
 hll_cardinality
-----
                2
(1 row)

-- Drop the table.
openGauss=# 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.
openGauss=# 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.
openGauss=# INSERT INTO facts VALUES ('2019-02-20', generate_series(1,100));
openGauss=# INSERT INTO facts VALUES ('2019-02-21', generate_series(1,200));
openGauss=# INSERT INTO facts VALUES ('2019-02-22', generate_series(1,300));
openGauss=# INSERT INTO facts VALUES ('2019-02-23', generate_series(1,400));
openGauss=# INSERT INTO facts VALUES ('2019-02-24', generate_series(1,500));
openGauss=# INSERT INTO facts VALUES ('2019-02-25', generate_series(1,600));
openGauss=# INSERT INTO facts VALUES ('2019-02-26', generate_series(1,700));
openGauss=# INSERT INTO facts VALUES ('2019-02-27', generate_series(1,800));

-- Create another table and specify an HLL column:
openGauss=# CREATE TABLE daily_uniques (
    date      date UNIQUE,
    users     hll
);

-- Group data by date and insert the data into the HLL.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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:
openGauss=# 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);
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.
openGauss=# DROP TABLE facts;
openGauss=# 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.

```
openGauss=# CREATE TABLE test(id integer, set hll);
openGauss=# INSERT INTO test VALUES(1, 'E\\1234');
ERROR: not a hll type, size=6 is not enough

openGauss=# 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 Range

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 a table and insert data into the table.
openGauss=# CREATE TABLE reservation (room int, during tsrange);
openGauss=# INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- Inclusion
openGauss=# SELECT int4range(10, 20) @> 3;
?column?
-----
f
(1 row)

-- Determine whether the two ranges overlap.
openGauss=# SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
?column?
-----
t
(1 row)

-- Extract the upper bound.
openGauss=# SELECT upper(int8range(15, 25));
upper
-----
25
(1 row)

-- Calculate the intersection.
openGauss=# SELECT int4range(10, 20) * int4range(15, 25);
?column?
-----
[15,20)
(1 row)

-- Determine whether the range is empty.
openGauss=# SELECT isempty(numrange(1, 5));
isempty
-----
f
(1 row)

-- Drop the table.
openGauss=# 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 two 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, the 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. 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 `""`. 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.
openGauss=# SELECT '[3,7)::int4range;
int4range
-----
[3,7)
(1 row)
-- All values between 3 (excluded) and 7 (excluded) are included.
openGauss=# SELECT '(3,7)::int4range;
int4range
-----
[4,7)
(1 row)
-- Only value 4 is included.
openGauss=# SELECT '[4,4)::int4range;
int4range
-----
[4,5)
(1 row)
-- No value is included (and will be normalized to null).
openGauss=# 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.
openGauss=# SELECT numrange(1.0, 14.0, '[]');
numrange
-----
(1.0,14.0]
(1 row)
-- If the third parameter is ignored, it is assumed to be '[]'.
openGauss=# 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.  
openGauss=# SELECT int8range(1, 14, '[');  
int8range  
-----  
[2,15)  
(1 row)  
-- Using NULL for a bound causes the range to be unbounded on that side.  
openGauss=# 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

B-tree and hash indexes 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 and hash support for range types is primarily designed to allow sorting and hashing 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. So, using a user-created table's **OID** column as a primary key is discouraged.

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 also 64-bit quantities.	N/A
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:


```
openGauss=# 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:

```
openGauss=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
attrelid | attname | atttypid | 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 | typtype | 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 | typtypmod | 23 | -1
1247 | typndims | 23 | -1
1247 | typcollation | 26 | -1
1247 | typdefaultbin | 194 | -1
1247 | typdefault | 25 | -1
1247 | typacl | 1034 | -1
(38 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.
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 or record as a result type. Some polymorphic functions support the anyelement, anyarray, anynonarray, anyenum, and anyrange types.

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 calling 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.  
openGauss=# create table t1 (a int);
```

```
-- Insert two data records.
openGauss=# insert into t1 values(1),(2);

-- Create the showall() function.
openGauss=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

-- Call the showall() function.
openGauss=# SELECT showall();
showall
-----
(2)
(1 row)

-- Delete the function.
openGauss=# DROP FUNCTION showall();

-- Drop the table.
openGauss=# drop table t1;
```

7.3.16 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 12-30](#).

Example:

```
-- Create users.
openGauss=# CREATE USER user1 WITH PASSWORD 'Aa123456789';
openGauss=# CREATE USER user2 WITH PASSWORD 'Aa123456789';
openGauss=# CREATE USER omm WITH PASSWORD 'Aa123456789';

-- Create a data table table_acl that contains three columns of the int, aclitem, and aclitem[] types.
openGauss=# 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.
openGauss=# 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.
openGauss=# INSERT INTO table_acl VALUES (2,'user1=aw/omm','{omm=d/user2}');
openGauss=# 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)

-- Drop the table and users.
openGauss=# DROP USER user1;
openGauss=# DROP USER user2;
openGauss=# DROP USER omm;
openGauss=# DROP TABLE table_acl;
```

7.4 Constant and Macro

[Table 7-24](#) lists the constants and macros that can be used in GaussDB.

Table 7-24 Constant and macro

Parameter	Description	Example
CURRENT_CATALOG	Specifies the current database.	testdb=# SELECT CURRENT_CATALOG; current_database ----- testdb (1 row)
CURRENT_ROLE	Specifies the current user.	openGauss=# SELECT CURRENT_ROLE; current_user ----- omm (1 row)
CURRENT_SCHEMA	Specifies the current database mode.	openGauss=# SELECT CURRENT_SCHEMA; current_schema ----- public (1 row)
CURRENT_USER	Specifies the current user.	openGauss=# SELECT CURRENT_USER; current_user ----- omm (1 row)
LOCALTIMESTAMP	Specifies the current session time (without time zone).	openGauss=# 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.	openGauss=# SELECT SESSION_USER; session_user ----- omm (1 row)
SYSDATE	Specifies the current system date.	openGauss=# SELECT SYSDATE; sysdate ----- 2015-10-10 15:48:53 (1 row)
USER	Specifies the current user, also called CURRENT_USER .	openGauss=# 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.

7.5.1 Logical Operators

Common logical operators include AND, OR, and NOT. The operation result has three values: **TRUE**, **FALSE**, and **NULL**. NULL indicates 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.

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

Operator	Description
=	Equal to
<>, !=, 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.

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:

```
openGauss=# 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:

```
openGauss=# SELECT btrim('sring', '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:

```
openGauss=# SELECT char_length('hello');
char_length
-----
5
(1 row)
```

- `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:

```
openGauss=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
instr
-----
6
(1 row)
```

- `lengthb(text/bpchar)`

Description: Obtains the number of bytes of a specified string.

Return type: int

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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

Example:

```
openGauss=# SELECT lpad('hi', 5, 'xyza');
lpad
-----
xyzhi
(1 row)
```

- **notlike(x bytea name text, y bytea text)**

Description: Compares x and y to check whether they are inconsistent.

Return type: Boolean

Example:

```
openGauss=# SELECT notlike(1,2);
notlike
-----
t
(1 row)
openGauss=# SELECT notlike(1,1);
notlike
-----
f
(1 row)
```

- **octet_length(string)**

Description: Specifies the number of bytes in a string.

Return type: int

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT position('ing' in 'string');
position
-----
4
(1 row)
```


- pg_client_encoding()

Description: Specifies the current client-side encoding name.

Return type: name

Example:

```
openGauss=# 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:

```
openGauss=# 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 an identifier in an SQL statement string (quotation marks are used as required).

Return type: text

Example:

```
openGauss=# SELECT quote_literal('hello');
quote_literal
-----
'hello'
(1 row)
```

If a command similar to the following exists, the text will be escaped.

```
openGauss=# 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.

```
openGauss=# 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**.

```
openGauss=# 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:

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# 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 literal in an SQL statement string (quotation marks are used as required).

Return type: text

Example:

```
openGauss=# SELECT quote_nullable('hello');
quote_nullable
-----
'hello'
(1 row)
```

If a command similar to the following exists, the text will be escaped.

```
openGauss=# 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.

```
openGauss=# SELECT quote_nullable('O\hello');
quote_nullable
-----
E'O\\hello'
(1 row)
```

If the parameter is null, **NULL** is returned.

```
openGauss=# 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:

```
openGauss=# 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.

```
openGauss=# 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.

```
openGauss=# SELECT quote_nullable('O\42.5');  
quote_nullable
```

```
-----  
E'O\\42.5'  
(1 row)
```

If the parameter is null, **NULL** is returned.

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT substring('Thomas' from 2 for 3);  
substring
```

```
-----  
hom  
(1 row)
```

- **substring(string from *pattern*)**

Description: Extracts substrings matching the POSIX 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:

```
openGauss=# SELECT substring('Thomas' from '...$');  
substring
```

```
-----  
mas  
(1 row)
```

```
openGauss=# SELECT substring('foobar' from 'o(.)b');  
result
```

```
-----  
o  
(1 row)  
openGauss=# SELECT substring('foobar' from '(o(.b)');  
result  
-----  
oob  
(1 row)
```

NOTE

If the POSIX pattern contains any parentheses, the portion of the text that matches 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:

```
openGauss=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');  
substring  
-----  
oma  
(1 row)
```

- `rawcat(raw,raw)`

Description: Indicates the string concatenation function.

Return type: raw

Example:

```
openGauss=# 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: Boolean

Example:

```
openGauss=# 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.

Parameter description:

- **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.
'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 ', the multi-line matching is used. Otherwise, the single-line matching is used.
'n'	The option meanings of "n" are related to the GUC parameter behavior_compat_options and the compatibility mode of the current database. • If the SQL compatibility mode of the database is A or B 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 '.

Option	Description
'x'	Indicates the loose matching. The whitespace characters are ignored.

Return type: text

Example:

```
openGauss=# SELECT regexp_substr('str','[ac]');
regexp_substr
-----
(1 row)

openGauss=# 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.
Parameter description:
 - **string**: source character string used for matching.
 - **pattern**: regular expression pattern string used for matching.
 - **position**: sequence number of the character to be matched from the source character string. 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](#).

Return type: int

Example:

```
openGauss=# 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.
Parameter description:
 - **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:

```
openGauss=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
-----
4
(1 row)

openGauss=# 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 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 function behavior. **i** indicates that the matching is not related to uppercase and lowercase. **g** indicates that each matching 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:

```
openGauss=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
regexp_matches
-----
{bar,beque}
(1 row)

openGauss=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
regexp_matches
-----
{barbeque}
```

```
(1 row)
openGauss=# SELECT regexp_matches('foobarbequebazilbarfbonk', '(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 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:

```
openGauss=# 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 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 contains zero or multiple single-letter flags that change the behavior of a function. **i** indicates that the matching is not related to uppercase and lowercase. By default, each matching substring is replaced, instead of replacing only the first one.

Return type: SETOF text

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT replace('abcdefabcdef', 'cd');
replace
-----
abefabef
(1 row)
```

- `reverse(str)`

Description: Returns the reversed string.

Return type: text

Example:

```
openGauss=# 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:

```
openGauss=# SELECT right('abcde', 2);
right
-----
de
(1 row)

openGauss=# SELECT right('abcde', -2);
right
-----
cde
(1 row)
```

- `rpad(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

Example:

```
openGauss=# SELECT rpad('hi', 5, 'xy');
rpad
-----
hixyx
(1 row)
```

- rtrim(string text [, characters text])

Description: Removes the longest string containing only characters from characters (a space by default) from the end of string.

Return type: text

Example:

```
openGauss=# SELECT rtrim('trimxxx', 'x');
rtrim
-----
trim
(1 row)
```

- 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT substr('string',2,3);
substr
-----
tri
(1 row)
```

- string || string

Description: Concatenates strings.

Return type: text

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT translate('12345', '143', 'ax');  
translate  
-----
```

```
a2x5  
(1 row)
```

- length(string)

Description: Obtains the number of characters in a string.

Return type: integer

Example:

```
openGauss=# SELECT length('abcd');  
length  
-----  
4  
(1 row)
```

- lengthb(string)

Description: Obtains the number of bytes in a string. The value depends on character sets (GBK and UTF8).

Return type: integer

Example:

```
openGauss=# 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, and *n* indicates the absolute value of **from**.

Return type: text

Example:

If the value of **from** is positive:

```
openGauss=# SELECT substr('ABCDEF',2);  
substr  
-----  
BCDEF  
(1 row)
```

If the value of **from** is negative:

```
openGauss=# 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:

```
openGauss=# SELECT substr('ABCDEF',2,2);
substr
-----
BC
(1 row)
```

If the value of **from** is negative:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT substrb('ABCDEF',2,2);
substrb
-----
BC
(1 row)
```

- 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:

```
openGauss=# SELECT trim(BOTH 'x' FROM 'xTomxx');
btrim
-----
Tom
(1 row)
openGauss=# SELECT trim(LEADING 'x' FROM 'xTomxx');
ltrim
```

```
-----  
Tomxx  
(1 row)  
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT ltrim('xxxxTRIM','x');  
ltrim  
-----  
TRIM  
(1 row)
```

- upper(string)

Description: Converts the string into the uppercase.

Return type: text

Example:

```
openGauss=# SELECT upper('tom');  
upper  
-----  
TOM  
(1 row)
```

- lower(string)

Description: Converts the string into the lowercase.

Return type: text

Example:

```
openGauss=# SELECT lower('TOM');  
lower  
-----  
tom  
(1 row)
```

- rpad(string varchar, length int [, fill varchar])

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.

length in GaussDB indicates the character length. One Chinese character is counted as one character.

Return type: text

Example:

```
openGauss=# SELECT rpad('hi',5,'xyza');
rpad
-----
hixyx
(1 row)
openGauss=# SELECT rpad('hi',5,'abcdefg');
rpad
-----
hiabc
(1 row)
```

- `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, the search is performed 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:

```
openGauss=# SELECT instr('corporate floor','or', 3);
instr
-----
5
(1 row)
openGauss=# 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:

```
openGauss=# SELECT initcap('hi THOMAS');
initcap
-----
Hi Thomas
(1 row)
```

- `ascii(string)`

Description: Indicates the ASCII code of the first character in the string.

Return type: integer

Example:

```
openGauss=# SELECT ascii('xyz');
ascii
-----
120
(1 row)
```

- `replace(string varchar, search_string varchar, replacement_string varchar)`

Description: Replaces all **search_string** in the string with **replacement_string**.

Return type: text

Example:

```
openGauss=# SELECT replace('jack and jue','j','bl');
replace
-----
black and blue
(1 row)
```

- `lpad(string varchar, length int[, repeat_string varchar])`

Description: Adds a series of **repeat_string** (a space by default) on the left of the string to generate a new string with the total length of *n*.

If the length of the string is longer than the specified length, the function truncates the string and returns the substrings with the specified length.

Return type: varchar

Example:

```
openGauss=# SELECT lpad('PAGE 1',15,'*');
lpad
-----
*****PAGE 1
(1 row)
openGauss=# SELECT lpad('hello world',5,'abcd');
lpad
-----
hello
(1 row)
```

- `concat(str1,str2)`

Description: Concatenates **str1** and **str2** and returns the concatenated string. If **str1** or **str2** is set to **NULL**, **NULL** is returned. Note: **concat** calls the output function of the data type and the return value is uncertain. 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 `||`.

NOTE

- 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.

Return type: text

Example:

```
openGauss=# SELECT concat('Hello', ' World!');
concat
-----
Hello World!
(1 row)
openGauss=# SELECT concat('Hello', NULL);
concat
-----
(1 row)
```

- `chr(integer)`

Description: Specifies the character of the ASCII code.

Return type: varchar

Example:

```
openGauss=# SELECT chr(65);
chr
----
```



```
A
(1 row)
```

- `regexp_replace(string, pattern, replacement [, flags ])`

Description: Replaces substrings matching the POSIX 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 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.

Return type: text

Example:

```
openGauss=# SELECT regexp_replace('Thomas', '[mN]a.', 'M');
regexp_replace
-----
ThM
(1 row)
openGauss=# SELECT regexp_replace('foobarbaz','b(.)', E'X\\1Y', 'g') AS
RESULT;
result
-----
fooXarYXazY
(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 A-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 **Postgres**.

Return type: text

Example:

```
openGauss=# 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:

```
openGauss=# create table test(a text);

openGauss=# insert into test(a) values('abC ');

openGauss=# insert into test(a) values('abC ');

openGauss=# insert into test(a) values('abc ');

openGauss=# select * from test order by nlssort(a,'nls_sort=schinese_pinyin_m');
 a
-----
abc
abC
abC
(3 rows)

openGauss=# select * from test order by nlssort(a, 'nls_sort=generic_m_ci');
 a
-----
abC
abc
abC
(3 rows)

openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# show server_encoding;
server_encoding
-----
LATIN1
(1 row)

openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
result
-----
t
(1 row)
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
result
-----
f
(1 row)
openGauss=# 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:

```
openGauss=# SELECT regexp_like('ABC', '[A-Z]');
regexp_like
-----
t
(1 row)
openGauss=# SELECT regexp_like('ABC', '[D-Z]');
regexp_like
-----
f
(1 row)
openGauss=# SELECT regexp_like('ABC', '[A-Z]', 'i');
regexp_like
-----
t
(1 row)
openGauss=# SELECT regexp_like('ABC', '[A-Z]');
regexp_like
-----
t
(1 row)
```

- **format**(formatstr text [, str"any" [, ...]])

Description: Formats a string.

Return type: text

Example:

```
openGauss=# SELECT format('Hello %s, %1$s', 'World');
format
-----
Hello World, World
(1 row)
```

- 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:

```
openGauss=# SELECT md5('ABC');
md5
-----
902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

- decode(string text, format text)

Description: Decodes binary data from textual representation.

Return type: bytea

Example:

```
openGauss=# 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:

```
openGauss=# select similar_escape('\s+ab','2');
similar_escape
-----
^(?!\s+ab)$
(1 row)
```

- encode(data bytea, format text)

Description: Encodes binary data into a textual representation.

Return type: text

Example:

```
openGauss=# 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 values of **length** and **lengthb** in GaussDB are 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, and the value can be **ORA**, **MYSQL**, **TD**, or **Postgres**. If the database type is not specified, **ORA** is used by default. In this case, the lexical analyzer is different from that of the other three databases, which considers an empty character string as **NULL**. Therefore, when **ORA** is specified, if an empty string is used as a parameter in the preceding character operation function, no output is displayed. For example:

```
openGauss=# 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. If **Postgres** is specified, the processing of character strings is the same as that of PostgreSQL. Therefore, the preceding problem does not occur.

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:

```
openGauss=# SELECT octet_length(E'jo\000se'::bytea) AS RESULT;
result
-----
5
(1 row)
```

- **overlay(string placing string from int [for int])**

Description: Replaces substrings.

Return type: bytea

Example:

```
openGauss=# SELECT overlay(E'Th\000omas'::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:

```
openGauss=# SELECT position(E'\000om'::bytea in E'Th\000omas'::bytea) AS RESULT;  
result  
-----  
3  
(1 row)
```

- `substring(string [from int] [for int])`

Description: Truncates substrings.

Return type: bytea

Example:

```
openGauss=# SELECT substring(E'Th\000omas'::bytea from 2 for 3) AS RESULT;  
result  
-----  
\x68006f  
(1 row)
```

- `substr(bytea [from int] [for int])`

Description: Truncates substrings.

Return type: bytea

Example:

```
openGauss=# select substr(E'Th\000omas'::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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT B'10001' || B'011' AS RESULT;
result
-----
10001011
(1 row)
```

NOTE

A column can have a maximum of 180 consecutive internal joins. A column with excessive joins will be split into joined consecutive strings.

Example: **str1||str2||str3||str4** is split into **(str1||str2)|| (str3||str4)**.

- **&**

Description: Specifies the AND operation between bit strings.

Example:

```
openGauss=# SELECT B'10001' & B'01101' AS RESULT;
result
-----
00001
(1 row)
```

- **|**

Description: Specifies the OR operation between bit strings.

Example:

```
openGauss=# 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:

```
openGauss=# SELECT B'10001' # B'01101' AS RESULT;
result
-----
11100
(1 row)
```

- **~**

Description: Specifies the NOT operation between bit strings.

Example:

```
openGauss=# SELECT ~B'10001' AS RESULT;
result
```

- ```

01110
(1 row)
```
- <<  
Description: Shifts left in a bit string.  
Example:  
openGauss=# SELECT B'10001' << 3 AS RESULT;  
result  
-----  
01000  
(1 row)
  - >>  
Description: Shifts right in a bit string.  
Example:  
openGauss=# 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:

```
openGauss=# SELECT 44::bit(10) AS RESULT;
result

0000101100
(1 row)

openGauss=# SELECT 44::bit(3) AS RESULT;
result

100
(1 row)

openGauss=# SELECT cast(-44 as bit(12)) AS RESULT;
result

111111010100
(1 row)

openGauss=# SELECT '1110'::bit(4)::integer AS RESULT;
result

14
(1 row)

openGauss=# 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, so you still need two backslashes. 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.

- The keyword ILIKE can be used instead of LIKE to make the match case-insensitive.
- Operator ~~ is equivalent to LIKE, and operator ~~\* corresponds to ILIKE.

Example:

```
openGauss=# SELECT 'abc' LIKE 'abc' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' LIKE 'a%' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' LIKE '_b_' AS RESULT;
result

t
(1 row)
```

```
openGauss=# 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:

- 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.
- The underscore (\_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.
- 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.                                         |
| +             | 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.                              |

- 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:

```
openGauss=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
result

f
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
result

t
(1 row)
openGauss=# 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 set 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 1 Regular expression match operators](#) lists all available operators for pattern matching using POSIX-style regular expressions.

**Table 7-28** 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:

- a. 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.
- b. 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 matching 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:**

```
openGauss=# SELECT 'abc' ~ 'Abc' AS RESULT;
result

f
(1 row)
openGauss=# SELECT 'abc' ~* 'Abc' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' !~ 'Abc' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' !~* 'Abc' AS RESULT;
result

f
(1 row)
openGauss=# SELECT 'abc' ~ '^a' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' ~ '(b|d)' AS RESULT;
result

t
(1 row)
openGauss=# SELECT 'abc' ~ '^ (b|c)' AS RESULT;
result

f
(1 row)
```

Although most regular expression searches can be performed quickly, regular expressions 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 Mathematical Functions and Operators

### Numeric Operators

- +

Description: Addition

**Example:**

```
openGauss=# SELECT 2+3 AS RESULT;
result

5
(1 row)
```

- -

Description: Subtraction



Example:

```
openGauss=# SELECT 2-3 AS RESULT;
result

-1
(1 row)
```

- \*

Description: Multiplication

Example:

```
openGauss=# SELECT 2*3 AS RESULT;
result

6
(1 row)
```

- /

Description: Division (The result is not rounded.)

Example:

```
openGauss=# SELECT 4/2 AS RESULT;
result

2
(1 row)
openGauss=# SELECT 4/3 AS RESULT;
result

1.3333333333333333
(1 row)
```

- +/-

Description: Positive/Negative

Example:

```
openGauss=# SELECT -2 AS RESULT;
result

-2
(1 row)
```

- %

Description: Modulo (to obtain the remainder)

Example:

```
openGauss=# SELECT 5%4 AS RESULT;
result

1
(1 row)
```

- @

Description: Absolute value

Example:

```
openGauss=# SELECT @ -5.0 AS RESULT;
result

5.0
(1 row)
```

- ^

Description: Power (exponent calculation)

Example:

```
openGauss=# SELECT 2.0^3.0 AS RESULT;
 result

8.0000000000000000
(1 row)
```

- $\sqrt{\quad}$

Description: Square root

Example:

```
openGauss=# SELECT $\sqrt{\quad}$ 25.0 AS RESULT;
 result

5
(1 row)
```

- $\sqrt[3]{\quad}$

Description: Cubic root

Example:

```
openGauss=# SELECT $\sqrt[3]{\quad}$ 27.0 AS RESULT;
 result

3
(1 row)
```

- $!$

Description: Factorial

Example:

```
openGauss=# SELECT 5! AS RESULT;
 result

120
(1 row)
```

- $!!$

Description: Factorial (prefix operator)

Example:

```
openGauss=# SELECT !!5 AS RESULT;
 result

120
(1 row)
```

- $\&$

Description: Binary AND

Example:

```
openGauss=# SELECT 91&15 AS RESULT;
 result

11
(1 row)
```

- $|$

Description: Binary OR

Example:

```
openGauss=# SELECT 32|3 AS RESULT;
 result

35
(1 row)
```

- #

Description: Binary XOR

Example:

```
openGauss=# SELECT 17#5 AS RESULT;
result

 20
(1 row)
```

- ~

Description: Binary NOT

Example:

```
openGauss=# SELECT ~1 AS RESULT;
result

 -2
(1 row)
```

- <<

Description: Binary shift left

Example:

```
openGauss=# SELECT 1<<4 AS RESULT;
result

 16
(1 row)
```

- >>

Description: Binary shift right

Example:

```
openGauss=# SELECT 8>>2 AS RESULT;
result

 2
(1 row)
```

## Numeric Operation Functions

- abs(x)

Description: Absolute value

Return type: same as the input

Example:

```
openGauss=# SELECT abs(-17.4);
abs

 17.4
(1 row)
```

- acos(x)

Description: Arc cosine

Return type: double precision

Example:

```
openGauss=# SELECT acos(-1);
acos

3.14159265358979
(1 row)
```

- **asin(x)**  
Description: Arc sine  
Return type: double precision  
Example:  

```
openGauss=# SELECT asin(0.5);
 asin

.523598775598299
(1 row)
```
- **atan(x)**  
Description: Arc tangent  
Return type: double precision  
Example:  

```
openGauss=# SELECT atan(1);
 atan

.785398163397448
(1 row)
```
- **atan2(y, x)**  
Description: Arc tangent of y/x  
Return type: double precision  
Example:  

```
openGauss=# SELECT atan2(2, 1);
 atan2

1.10714871779409
(1 row)
```
- **bitand(integer, integer)**  
Description: Performs an AND (&) operation on two integers.  
Return type: bigint  
Example:  

```
openGauss=# SELECT bitand(127, 63);
 bitand

63
(1 row)
```
- **cbrt(dp)**  
Description: Cubic root  
Return type: double precision  
Example:  

```
openGauss=# SELECT cbrt(27.0);
 cbrt

3
(1 row)
```
- **ceil(x)**  
Description: Minimum integer greater than or equal to the parameter  
Return type: integer  
Example:  

```
openGauss=# 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:

```
openGauss=# SELECT ceiling(-95.3);
ceiling

-95
(1 row)
```

- **cos(x)**

Description: Cosine

Return type: double precision

Example:

```
openGauss=# SELECT cos(-3.1415927);
cos

-.9999999999999999
(1 row)
```

- **cot(x)**

Description: Cotangent

Return type: double precision

Example:

```
openGauss=# SELECT cot(1);
cot

.642092615934331
(1 row)
```

- **degrees(dp)**

Description: Converts radians to angles.

Return type: double precision

Example:

```
openGauss=# SELECT degrees(0.5);
degrees

28.6478897565412
(1 row)
```

- **div(y numeric, x numeric)**

Description: Integer part of y/x

Return type: numeric

Example:

```
openGauss=# SELECT div(9,4);
div

2
(1 row)
```

- **exp(x)**

Description: Natural exponent

Return type: same as the input

Example:

```
openGauss=# SELECT exp(1.0);
exp

2.7182818284590452
(1 row)
```

- floor(x)

Description: Maximum integer not larger than the parameter

Return type: same as the input

Example:

```
openGauss=# 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:

```
openGauss=# select int1('123');
int1

123
(1 row)
openGauss=# select int1('a');
int1

0
(1 row)
```

- 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:

```
openGauss=# select int2('1234');
int2

1234
(1 row)
openGauss=# select int2(25.3);
int2

25
(1 row)
```

- 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:

```
openGauss=# select int4('789');
int4

789
(1 row)
openGauss=# select int4(99.9);
int4

99
(1 row)
```

- **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:

```
openGauss=# select int8('789');
int8

789
(1 row)
openGauss=# select int8(99.9);
int8

99
(1 row)
```

- **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:

```
openGauss=# select float4('789');
float4

789
(1 row)
openGauss=# 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:

```
openGauss=# select float8('789');
float8

789
(1 row)
openGauss=# 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:

```
openGauss=# select int16('789');
int16

 789
(1 row)
```

```
openGauss=# select int16(99.9);
int16

 99
(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:

```
openGauss=# select "numeric"('789');
numeric

 789
(1 row)
```

```
openGauss=# 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:

```
openGauss=# SELECT radians(45.0);
radians

.785398163397448
(1 row)
```

- **random()**

Description: Random number between 0.0 and 1.0

Return type: double precision



Example:

```
openGauss=# 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:

```
openGauss=# SELECT multiply(9.0, '3.0');
multiply

27
(1 row)
openGauss=# SELECT multiply('9.0', 3.0);
multiply

27
(1 row)
```

- ln(x)

Description: Natural logarithm

Return type: same as the input

Example:

```
openGauss=# SELECT ln(2.0);
ln

.6931471805599453
(1 row)
```

- log(x)

Description: Logarithm with 10 as the base

Return type: same as the input

Example:

```
openGauss=# 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:

```
openGauss=# SELECT log(2.0, 64.0);
log

6.0000000000000000
(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:

```
openGauss=# SELECT mod(9,4);
mod

1
(1 row)
openGauss=# SELECT mod(9,0);
mod

9
(1 row)
```

- **pi()**

Description:  $\pi$  constant value

Return type: double precision

Example:

```
openGauss=# SELECT pi();
pi

3.14159265358979
(1 row)
```

- **power(a double precision, b double precision)**

Description: b power of a

Return type: double precision

Example:

```
openGauss=# SELECT power(9.0, 3.0);
power

729.0000000000000000
(1 row)
```

- **round(x)**

Description: Integer closest to the input parameter

Return type: same as the input

Example:

```
openGauss=# SELECT round(42.4);
round

42
(1 row)
openGauss=# SELECT round(42.6);
round

43
(1 row)
```

- **round(v numeric, s int)**

Description: s digits are kept after the decimal point.

Return type: numeric

Example:

```
openGauss=# SELECT round(42.4382, 2);
round

42.44
(1 row)
```

- **setseed(dp)**

Description: Sets seed for the following random() calling (between -1.0 and 1.0, inclusive).

Return type: void

Example:

```
openGauss=# SELECT setseed(0.54823);
setseed

(1 row)
```

- **sign(x)**

Description: Returns symbols of this parameter.

Return type: **-1** indicates minus. **0** indicates 0, and **1** indicates positive numbers.

Example:

```
openGauss=# SELECT sign(-8.4);
sign

-1
(1 row)
```

- **sin(x)**

Description: Sine

Return type: double precision

Example:

```
openGauss=# SELECT sin(1.57079);
sin

.999999999979986
(1 row)
```

- **sqrt(x)**

Description: Square root

Return type: same as the input

Example:

```
openGauss=# SELECT sqrt(2.0);
sqrt

1.414213562373095
(1 row)
```

- **tan(x)**

Description: Tangent

Return type: double precision

Example:

```
openGauss=# SELECT tan(20);
tan

2.23716094422474
(1 row)
```

- **trunc(x)**

Description: Truncates (the integral part).

Return type: same as the input

Example:

```
openGauss=# 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:

```
openGauss=# SELECT trunc(42.4382, 2);
trunc

 42.43
(1 row)
```

- `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:

```
openGauss=# 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:

```
openGauss=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
width_bucket

 3
(1 row)
```

- `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 a 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 MOD operation on unsigned 1-byte integers.  
Parameter: tinyint, tinyint  
Return type: tinyint
- **int1mul**  
Description: Performs a multiplication operation on 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

## 7.5.8 Date and Time Processing Functions and Operators

### Date and Time Operators



#### WARNING

When the user uses date/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.

```
openGauss=# SELECT date '2001-10-01' - '7' AS RESULT;
ERROR:
GAUSS-10416: invalid input syntax for type timestamp: "7"
SQLSTATE: 22007
LINE 1: SELECT date '2001-10-01' - '7' AS RESULT;
 ^
CONTEXT: referenced column: result
```

**Table 7-29** Time and date operators

| Operator | Example                                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------------------------------|
| +        | <pre>openGauss=# SELECT date '2001-9-28' + integer '7' AS RESULT;<br/>result<br/>-----<br/>2001-10-05<br/>(1 row)</pre> |

| Operator | Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | <p>openGauss=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT;<br/>result<br/>-----<br/>2001-09-28 01:00:00<br/>(1 row)</p> <p>openGauss=# SELECT date '2001-09-28' + time '03:00' AS RESULT;<br/>result<br/>-----<br/>2001-09-28 03:00:00<br/>(1 row)</p> <p>openGauss=# SELECT interval '1 day' + interval '1 hour' AS RESULT;<br/>result<br/>-----<br/>1 day 01:00:00<br/>(1 row)</p> <p>openGauss=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT;<br/>result<br/>-----<br/>2001-09-29 00:00:00<br/>(1 row)</p> <p>openGauss=# SELECT time '01:00' + interval '3 hours' AS RESULT;<br/>result<br/>-----<br/>04:00:00<br/>(1 row)</p>                                                                                                   |
| -        | <p>openGauss=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT;<br/>result<br/>-----<br/>3<br/>(1 row)</p> <p>openGauss=# SELECT date '2001-10-01' - integer '7' AS RESULT;<br/>result<br/>-----<br/>2001-09-24 00:00:00<br/>(1 row)</p> <p>openGauss=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT;<br/>result<br/>-----<br/>2001-09-27 23:00:00<br/>(1 row)</p> <p>openGauss=# SELECT time '05:00' - time '03:00' AS RESULT;<br/>result<br/>-----<br/>02:00:00<br/>(1 row)</p> <p>openGauss=# SELECT time '05:00' - interval '2 hours' AS RESULT;<br/>result<br/>-----<br/>03:00:00<br/>(1 row)</p> <p>openGauss=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT;<br/>result<br/>-----<br/>2001-09-28 00:00:00<br/>(1 row)</p> |



| Operator | Example                                                                                                                                                                                                                                                                                                        |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | <pre>openGauss=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ----- 23:00:00 (1 row)</pre> <pre>openGauss=# 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>openGauss=# SELECT 900 * interval '1 second' AS RESULT; result ----- 00:15:00 (1 row)</pre> <pre>openGauss=# SELECT 21 * interval '1 day' AS RESULT; result ----- 21 days (1 row)</pre> <pre>openGauss=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result ----- 03:30:00 (1 row)</pre> |
| /        | <pre>openGauss=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ----- 00:40:00 (1 row)</pre>                                                                                                                                                                                              |

## Time/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:

```
openGauss=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
age

43 years 9 mons 27 days
(1 row)
```

- **age(timestamp)**  
Description: Minuses the current time with the parameter. The input parameter can contain timezone or not.

Return type: interval

Example:

```
openGauss=# SELECT age(timestamp '1957-06-13');
age
```

```

60 years 2 mons 18 days
(1 row)
```

- `clock_timestamp()`

Description: Specifies the current timestamp of the real-time clock. 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:

```
openGauss=# SELECT clock_timestamp();
clock_timestamp

2017-09-01 16:57:36.636205+08
(1 row)
```

- `current_date`

Description: Specifies the current date.

Return type: date

Example:

```
openGauss=# SELECT current_date;
date

2017-09-01
(1 row)
```

- `current_time`

Description: Specifies the current time.

Return type: time with time zone

Example:

```
openGauss=# SELECT current_time;
timetz

16:58:07.086215+08
(1 row)
```

- `current_timestamp`

Description: Specifies the current date and time. This is a statement-level timestamp. The returned results within the same statement remain unchanged.

Return type: timestamp with time zone

Example:

```
openGauss=# SELECT current_timestamp;
pg_systimestamp

2017-09-01 16:58:19.22173+08
(1 row)
```

- `pg_systimestamp()`

Description: Current date and time (start of the current statement).

Return type: timestamp with time zone

Example:

```
openGauss=# SELECT pg_systimestamp();
pg_systimestamp

2015-10-14 11:21:28.317367+08
(1 row)
```

- `date_part(text, timestamp)`

Description:

Retrieves subcolumns 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:

```
openGauss=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
date_part

 20
(1 row)
```

- `timestamp_diff(text, timestamp, timestamp)`

Description: Calculates the difference between two timestamps and truncates the difference to the precision specified by text. In MY-compatible database, this function is the same as **`TIMESTAMPDIFF(unit, timestamp_expr1, timestamp_expr2)`**.

Return type: int64

Example:

```
openGauss=# SELECT timestamp_diff('year','2018-01-01','2020-04-01');
timestamp_diff

 2
(1 row)
openGauss=# SELECT timestamp_diff('month','2018-01-01','2020-04-01');
timestamp_diff

 27
(1 row)
openGauss=# SELECT timestamp_diff('quarter','2018-01-01','2020-04-01');
timestamp_diff

 9
(1 row)
openGauss=# SELECT timestamp_diff('week','2018-01-01','2020-04-01');
timestamp_diff

 117
(1 row)
openGauss=# SELECT timestamp_diff('day','2018-01-01','2020-04-01');
timestamp_diff

 821
(1 row)
openGauss=# SELECT timestamp_diff('hour','2018-01-01 10:10:10','2018-01-01 12:12:12');
timestamp_diff

 2
(1 row)
openGauss=# SELECT timestamp_diff('minute','2018-01-01 10:10:10','2018-01-01 12:12:12');
timestamp_diff

 122
(1 row)
openGauss=# SELECT timestamp_diff('second','2018-01-01 10:10:10','2018-01-01 10:12:12');
timestamp_diff

 122
(1 row)
```

```
openGauss=# SELECT timestamp_diff('microsecond','2018-01-01 10:10:10','2018-01-01 10:12:12');
timestamp_diff

122000000
(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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT trunc(timestamp '2001-02-16 20:38:40',
'hour');
trunc

2001-02-16 20:00:00
(1 row)
```

- `daterange(arg1, arg2)`

Description: Obtains time boundary information.

**arg1** type: date

**arg2** type: date

Return type: daterange

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT extract(month from interval '2 years 3 months');
 date_part

3
(1 row)
```

- **isfinite(date)**

Description: Tests for a valid date.

Return type: Boolean

Example:

```
openGauss=# SELECT isfinite(date '2001-02-16');
 isfinite

t
(1 row)
```

- **isfinite(timestamp)**

Description: Tests for a valid timestamp.

Return type: Boolean

Example:

```
openGauss=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
isfinite

t
(1 row)
```

- **isfinite(interval)**

Description: Tests for a valid interval.

Return type: Boolean

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
justify_interval

29 days 23:00:00
(1 row)
```

- **localtime**

Description: Specifies the current time.

Return type: time

Example:

```
openGauss=# SELECT localtime AS RESULT;
result

```

```
16:05:55.664681
(1 row)
```

- **localtimestamp**

Description: Specifies the current date and time.

Return type: timestamp

Example:

```
openGauss=# SELECT localtimestamp;
 timestamp

2017-09-01 17:03:30.781902
(1 row)
```

- **now()**

Description: Specifies the current date and time. This is a transaction-level timestamp. The results returned within the same transaction remain unchanged.

Return type: timestamp with time zone

Example:

```
openGauss=# SELECT now();
 now

2017-09-01 17:03:42.549426+08
(1 row)
```

- **timenow()**

Description: Specifies the current date and time.

Return type: timestamp with time zone

Example:

```
openGauss=# 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 **oracle** to be compatible with the interval output format of the function in the Oracle Database.

Example:

```
openGauss=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval

100:00:00
(1 row)

openGauss=# SET intervalstyle = oracle;
SET
openGauss=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval

+0000000004 04:00:00.000000000
(1 row)
```

- **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:

```
openGauss=# 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:

```
openGauss=# SELECT statement_timestamp();
statement_timestamp

2017-09-01 17:04:39.119267+08
(1 row)
```

- **sysdate**

Description: Specifies the current date and time.

Return type: timestamp

Example:

```
openGauss=# SELECT sysdate;
sysdate

2017-09-01 17:04:49
(1 row)
```

- **timeofday()**

Description: Specifies the current date and time (like **clock\_timestamp**, but returned as a **text** string)

Return type: text

Example:

```
openGauss=# SELECT timeofday();
timeofday

Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- **transaction\_timestamp()**

Description: Specifies the current date and time (equivalent to **current\_timestamp**)

Return type: timestamp with time zone

Example:

```
openGauss=# SELECT transaction_timestamp();
transaction_timestamp

2017-09-01 17:05:13.534454+08
(1 row)
```

- **add\_months(d,n)**

Description: Returns the time point *d* plus *n* 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:

```
openGauss=# 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)
```

- **last\_day(d)**

Description: Returns the date of the last day of the month that contains time point *d*.

Return type: timestamp

Example:

```
openGauss=# select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
cal_result

2017-01-31 00:00:00
(1 row)
```

- **next\_day(x,y)**

Description: Calculates the time of the next week *y* started from *x*.

Return type: timestamp

Example:

```
openGauss=# select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
cal_result

2017-05-28 00:00:00
(1 row)
```

- **tinterval(abstime, abstime)**

Description: Creates a time interval with two pieces of absolute time.

Return type: tinterval

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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. This can be considered as 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 7-30** shows the templates for truncating date/time values.

**Table 7-30** Truncating date/time values

| Item        | Format     | Description                                                              |
|-------------|------------|--------------------------------------------------------------------------|
| Microsecond | MICROSECON | Truncates date/time values, accurate to the microsecond (000000–999999). |
|             | US         |                                                                          |
|             | USEC       |                                                                          |

| Item        | Format     | Description                                                                                 |
|-------------|------------|---------------------------------------------------------------------------------------------|
|             | USECOND    |                                                                                             |
| 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    |                                                                                             |

| Item       | Format     | Description                                                                                       |
|------------|------------|---------------------------------------------------------------------------------------------------|
| Year       | Y          | Truncates date/time values, accurate to the year (the first day of the current year).             |
|            | YEAR       |                                                                                                   |
|            | 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 |                                                                                                   |

## 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, timestampz, or date type. **unit** specifies the unit of the difference between two dates.

### NOTE

This function is valid only when GaussDB is compatible with MySQL (that is, **dbcompatibility** is set to 'MYSQL'). It is equivalent to **timestamp\_diff(text, timestamp, timestamp)**.

- year

Year.

```
openGauss=# SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
timestamp_diff

2
(1 row)
```

- quarter

Quarter.

```
openGauss=# SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
timestamp_diff

8
(1 row)
```

- month

Month.

```
openGauss=# SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
timestamp_diff

 24
(1 row)
```

- week

Week.

```
openGauss=# SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
timestamp_diff

 104
(1 row)
```

- day

Day.

```
openGauss=# SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
timestamp_diff

 730
(1 row)
```

- hour

Hour.

```
openGauss=# SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff

 1
(1 row)
```

- minute

Minute.

```
openGauss=# SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff

 61
(1 row)
```

- second

Second.

```
openGauss=# SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
timestamp_diff

 3661
(1 row)
```

- microseconds

The seconds column, including fractional parts, is multiplied by 1,000,000.

```
openGauss=# SELECT TIMESTAMPDIFF(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

```
openGauss=# SELECT TIMESTAMPDIFF(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 subcolumns 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 column 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 numbered 0. The century goes from the 1st century BC to the 1st century AD.

Example:

```
openGauss=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
date_part

 20
(1 row)
```

- **day**

- Specifies the day (1–31) of the month for **timestamp**.

```
openGauss=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

 16
(1 row)
```

- Specifies the number of days for **interval**.

```
openGauss=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
date_part

 40
(1 row)
```

- **decade**

Specifies the number of decades.

```
openGauss=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

 200
(1 row)
```

- **dow**

Specifies the day of the week. The value range from 0 (Sunday) to 6 (Saturday).

```
openGauss=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

 5
(1 row)
```

- **doy**

Specifies the day of the year (1–365 or 1–366).

```
openGauss=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
```



```

47
(1 row)
```

- **epoch**

- Specifies the number of seconds since 1970-01-01 00:00:00-00 UTC (can be negative) for **timestamp with time zone** values.

Specifies the number of seconds since 1970-01-01 00:00:00-00 local time for **date** and **timestamp** values.

Specifies the total number of seconds in the interval for **interval** values.

```
openGauss=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
20:38:40.12-08');
date_part
```

```

982384720.12
(1 row)
```

```
openGauss=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
date_part
```

```

442800
(1 row)
```

- The following converts an epoch value back to a timestamp.

```
openGauss=# 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**

Specifies the hour column (0–23).

```
openGauss=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
```

```

20
(1 row)
```

- **isodow**

Specifies the day of the week (1–7).

**1** indicates Monday and **7** indicates Sunday.

 **NOTE**

This is identical to **dow** except for Sunday.

```
openGauss=# 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 the **week** column for more information.

```
openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
date_part
```

```

2005
(1 row)
```

```
openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
date_part
```

```

2006
(1 row)
```

- **microseconds**

The seconds column, including fractional parts, is multiplied by 1,000,000.

```
openGauss=# 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.

```
openGauss=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

3
(1 row)
```

- **milliseconds**

Seconds column, including fractional parts, is multiplied by 1000. Note that this includes full seconds.

```
openGauss=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
date_part

28500
(1 row)
```

- **minute**

Specifies the minute column (0–59).

```
openGauss=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

38
(1 row)
```

- **month**

Specifies the month column (1–12) for **timestamp** values.

```
openGauss=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

2
(1 row)
```

Specifies the number of months after modulo 12 is performed (0–11) for **interval** values.

```
openGauss=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
date_part

1
(1 row)
```

- **quarter**

Specifies the quarter of the year (1–4) that the date is in.

```
openGauss=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

1
(1 row)
```

- **second**

Specifies the second column, including fractional parts (0–59).

```
openGauss=# 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, **2005-01-01** is part of the 53rd week of year 2004, **2006-01-01** is part of the 52nd week of year 2005, and **2012-12-31** is part of the 1st week of year 2013. You are advised to use the columns **isoyear** and **week** together to ensure consistency.

```
openGauss=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
date_part

 7
(1 row)
```

- **year**

Specifies the year column.

```
openGauss=# 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 are the same as those for **extract**. For details, see [EXTRACT](#).

Example:

```
openGauss=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
date_part

 16
(1 row)
```

```
openGauss=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
date_part

 4
(1 row)
```

**Table 7-31** specifies the schema for formatting date and time values.

**Table 7-31** 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) |
|                       | YYYY       | 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              |

| Category    | Format                                                                            | Description                                                                                                                                                                                 |
|-------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | 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. |
|             | <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 name 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 name 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                                                                                                                                                                                     |

 **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 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.

## 7.5.9 Type Conversion Functions

### Type Conversion Functions

- `cash_words(money)`

Description: Type conversion function, which converts money into text.

Example:

```
openGauss=# SELECT cash_words('1.23');
 cash_words

One dollar and twenty three cents
(1 row)
```

- `cast(x as y)`

Description: Converts x into the type specified by y.

Example:

```
openGauss=# SELECT cast('22-oct-1997' as timestamp);
 timestamp

1997-10-22 00:00:00
(1 row)
```

- `hextoraw(text)`

Description: Converts a string in hexadecimal format into raw type.

Return type: raw

Example:

```
openGauss=# 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:

```
openGauss=# SELECT numtoday(2);
 numtoday

2 days
(1 row)
```

- rawtohex(string)

Description: Converts a string in binary format into hexadecimal format.  
The result is the ASCII code of the input characters in hexadecimal format.

Return type: varchar

Example:

```
openGauss=# SELECT rawtohex('1234567');
 rawtohex

31323334353637
(1 row)
```

- to\_bigint(varchar)

Description: Converts the character type to the bigint type.

Return type: bigint

Example:

```
openGauss=# SELECT to_bigint('123364545554455');
 to_bigint

123364545554455
(1 row)
```

- 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 VARCHAR 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.
- 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:

```
openGauss=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char

10:19:26
(1 row)
openGauss=# 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:

```
openGauss=# 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 characters, grouping characters, 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 (.).
  - A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the **template0**.
  - Use the template X or x to convert an integer value into a string in hexadecimal format.

Return type: text

Example:

```
openGauss=# SELECT to_char(1485,'9,999');
to_char

1,485
(1 row)
openGauss=# SELECT to_char(1148.5,'9,999.999');
to_char

1,148.500
(1 row)
openGauss=# SELECT to_char(148.5,'990999.909');
to_char

0148.500
(1 row)
openGauss=# SELECT to_char(123,'XXX');
to_char

7B
(1 row)
```

- `to_char(interval, text)`  
Description: Converts the values of the time interval type into the strings in the specified format.

Return type: text

Example:

```
openGauss=# 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:

```
openGauss=# SELECT to_char(125, '999');
to_char

125
(1 row)
```



- `to_char(numeric, text)`

Description: Converts the values of the numeric type into the strings in the specified format.

Return type: text

Example:

```
openGauss=# 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 VARCHAR 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:

```
openGauss=# SELECT to_char('01110');
to_char

01110
(1 row)
```

- `to_nvarchar2`

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:

```
openGauss=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
to_char

10:55:59
(1 row)
```

#### NOTE

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_clob(char/nchar/varchar/nvarchar/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:

```
openGauss=# SELECT to_clob('ABCDEF'::RAW(10));
to_clob

ABCDEF
(1 row)
```

```
openGauss=# SELECT to_clob('hello111'::CHAR(15));
to_clob

hello111
(1 row)
openGauss=# SELECT to_clob('gauss123'::NCHAR(10));
to_clob

gauss123
(1 row)
openGauss=# SELECT to_clob('gauss234'::VARCHAR(10));
to_clob

gauss234
(1 row)
openGauss=# SELECT to_clob('gauss345'::VARCHAR2(10));
to_clob

gauss345
(1 row)
openGauss=# SELECT to_clob('gauss456'::NVARCHAR2(10));
to_clob

gauss456
(1 row)
openGauss=# 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.

- 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:

```
openGauss=# 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:

```
openGauss=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
to_date

2000-12-05 00:00:00
(1 row)
```

- **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 7-32](#).

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 a sign-free 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:

```
openGauss=# 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:

```
openGauss=# SELECT to_number('12,454.8-', '99G999D9S');
to_number

-12454.8
(1 row)
```

- `to_timestamp(double precision)`

Description: Converts a Unix century into a timestamp.

Return type: timestamp with time zone

Example:

```
openGauss=# 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 *SY* 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 date and time. Otherwise, an error is reported.

Return type: timestamp without time zone

Example:

```
openGauss=# SHOW nls_timestamp_format;
nls_timestamp_format

DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

openGauss=# SELECT to_timestamp('12-sep-2014');
```

```
to_timestamp

2014-09-12 00:00:00
(1 row)
openGauss=# 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)
openGauss=# SELECT to_timestamp('-1','YYYY');
to_timestamp

0001-01-01 00:00:00 BC
(1 row)
openGauss=# SELECT to_timestamp('98','RR');
to_timestamp

1998-01-01 00:00:00
(1 row)
openGauss=# SELECT to_timestamp('01','RR');
to_timestamp

2001-01-01 00:00:00
(1 row)
```

- `to_timestamp(text, text)`

Description: Converts values of the string type into the timestamp of the specified type.

Return type: timestamp

Example:

```
openGauss=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
to_timestamp

2000-12-05 00:00:00
(1 row)
```

**Table 7-32** Template patterns for numeric formatting

| Format     | Description                                                         |
|------------|---------------------------------------------------------------------|
| 9          | Value with specified digits                                         |
| 0          | Values 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) |

| Format   | Description                                                           |
|----------|-----------------------------------------------------------------------|
| 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                          |

- `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 bool to int1.  
Parameter: Boolean  
Return type: tinyint
- `bool_int2`  
Description: Converts bool to int2.  
Parameter: Boolean  
Return type: smallint
- `bool_int8`  
Description: Converts bool 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 bool.  
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 signed two-byte integer to the bool 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 an eight-byte signed integer to a Boolean value.  
Parameter: `bigint`  
Return type: `Boolean`
- `int8_bpchar`  
Description: Converts an 8-byte signed integer to bpchar.  
Parameter: `bigint`  
Return type: `character`
- `int8_text`  
Description: Converts an eight-byte signed integer to the text type.  
Parameter: `bigint`  
Return type: `text`
- `int8_varchar`  
Description: Converts an eight-byte signed integer to varchar.  
Parameter: `bigint`  
Return type: `character varying`

- `intervaltonum`  
Description: Converts the internal date type 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`

- `oidvector_send_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
- `timestamptz_to_smalldatetime`  
Description: Converts timestamptz 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

## Encoding Type Conversion

- `convert_to_nocase(text, text)`  
Description: Converts a string into a specified encoding type.  
Return type: bytea  
Example:

```
openGauss=# SELECT convert_to_nocase('12345', 'GBK');
convert_to_nocase

\x3132333435
(1 row)
```

## 7.5.10 Geometric Functions and Operators

### Geometric Operators

- `+`  
Description: Translation  
Example:  

```
openGauss=# SELECT box '((0,0),(1,1))' + point '(2,0,0)' AS RESULT;
result

(3,1),(2,0)
(1 row)
```
- `-e`  
Description: Translation  
Example:  

```
openGauss=# SELECT box '((0,0),(1,1))' - point '(2,0,0)' AS RESULT;
result

(-1,1),(-2,0)
(1 row)
```

- \*

Description: Scaling out/Rotation

Example:

```
openGauss=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
result

(2,2),(0,0)
(1 row)
```
- /

Description: Scaling in/Rotation

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT # path '((1,0),(0,1),(-1,0))' AS RESULT;
result

3
(1 row)
```
- @-@

Description: Length or circumference

Example:

```
openGauss=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
result

2
(1 row)
```
- @@

Description: Center of box

Example:

```
openGauss=# SELECT @@ circle '((0,0),10)' AS RESULT;
result

(0,0)
(1 row)
```
- <->

Description: Distance between the two figures

Example:

```
openGauss=# 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:

```

openGauss=# 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:

```

openGauss=# 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:

```

openGauss=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
result
-----
t
(1 row)

```
- **&<**

Description: Does not extend to the right of?

Example:

```

openGauss=# 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:

```

openGauss=# 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:

```

openGauss=# 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:

```
openGauss=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
result
-----
t
(1 row)
```

- &<|

Description: Does not extend above?

Example:

```
openGauss=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- |&>

Description: Does not extend below?

Example:

```
openGauss=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- <^

Description: Is below (allows touching)?

Example:

```
openGauss=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- >^

Description: Is above (allows touching)?

Example:

```
openGauss=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))' AS RESULT;
result
-----
t
(1 row)
```

- ?#

Description: Intersect?

Example:

```
openGauss=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

- ?-

Description: Is horizontal?

Example:

```
openGauss=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
result
-----
t
(1 row)
```

- ?-

Description: Are horizontally aligned?

Example:

```
openGauss=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
result
-----
t
(1 row)
```

- ?|

Description: Is vertical?

Example:

```
openGauss=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
result
-----
f
(1 row)
```

- ?|

Description: Are vertically aligned?

Example:

```
openGauss=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
result
-----
t
(1 row)
```

- ?-|

Description: Are perpendicular?

Example:

```
openGauss=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
result
-----
t
(1 row)
```

- ?||

Description: Are parallel?

Example:

```
openGauss=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
result
-----
t
(1 row)
```

- @>

Description: Contains?

Example:

```
openGauss=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
result
-----
t
(1 row)
```

- <@

Description: Contained in or on?

Example:

```
openGauss=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
result
```

- ```

t
(1 row)
```
- `~=`  
Description: Same as?  
Example:  

```
openGauss=# 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:  

```
openGauss=# SELECT area(box '((0,0),(1,1))') AS RESULT;
result

1
(1 row)
```
- `center(object)`  
Description: Figure center calculation  
Return type: point  
Example:  

```
openGauss=# 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:  

```
openGauss=# 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:  

```
openGauss=# SELECT height(box '((0,0),(1,1))') AS RESULT;
result

1
(1 row)
```
- `isclosed(path)`  
Description: A closed path?

Return type: Boolean

Example:

```
openGauss=# SELECTisclosed(path '((0,0),(1,1),(2,0)))' AS RESULT;
result

t
(1 row)
```

- **isopen(path)**

Description: An open path?

Return type: Boolean

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
result

2
(1 row)
```

- **pclose(path)**

Description: Converts path to closed.

Return type: path

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT circle(polygon '((0,0),(1,1),(2,0))) AS RESULT;
result

<(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

Description: Box diagonal to line segment

Return type: lseg

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
result

1
(1 row)
```

- **path(polygon)**  
Description: Polygon to path  
Return type: path

Example:

```
openGauss=# 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: node  
Return type: point

Example:

```
openGauss=# 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:

```
openGauss=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
result

(0,0)
(1 row)
```

- **point(circle)**  
Description: Center of circle  
Return type: point

Example:

```
openGauss=# 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:

```
openGauss=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
result
```



- ```
-----  
(0,0)  
(1 row)
```
- **point(polygon)**
Description: Center of polygon
Return type: point
Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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,-0.9999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),  
(-0.9999999999999999,-1.73205080756888),(-1.73205080756888,-1))  
(1 row)
```
- **polygon(npts, circle)**
Description: Circle to **npts**-point polygon
Return type: polygon
Example:

```
openGauss=# 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,-0.9999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),  
(-0.9999999999999999,-1.73205080756888),(-1.73205080756888,-1))  
(1 row)
```
- **polygon(path)**
Description: Path to polygon

Return type: polygon

Example:

```
openGauss=# 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:

```
openGauss=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;  
result  
-----  
t  
(1 row)
```

- <=

Description: Is less than or equals

Example:

```
openGauss=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

- =

Description: Equals

Example:

```
openGauss=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

- >=

Description: Is greater than or equals

Example:

```
openGauss=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

- >

Description: Is greater than

Example:

```
openGauss=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;  
result
```

- ```

t
(1 row)
```
- <>

Description: Does not equal to

Example:

```
openGauss=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
result

t
(1 row)
```
- <<

Description: Is contained in

Example:

```
openGauss=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
result

t
(1 row)
```
- <<=

Description: Is contained in or equals

Example:

```
openGauss=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
result

t
(1 row)
```
- >>

Description: Contains

Example:

```
openGauss=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
result

t
(1 row)
```
- >>=

Description: Contains or equals

Example:

```
openGauss=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
result

t
(1 row)
```
- ~

Description: Bitwise NOT

Example:

```
openGauss=# SELECT ~ inet '192.168.1.6' AS RESULT;
result

63.87.254.249
(1 row)
```
- &

Description: Performs an AND operation on each bit of the two network addresses.

Example:

```
openGauss=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
result

0.0.0.0
(1 row)
```

- |

Description: Performs an OR operation on each bit of the two network addresses.

Example:

```
openGauss=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
result

202.168.1.6
(1 row)
```

- +

Description: Addition

Example:

```
openGauss=# SELECT inet '192.168.1.6' + 25 AS RESULT;
result

192.168.1.31
(1 row)
```

- -

Description: Subtraction

Example:

```
openGauss=# SELECT inet '192.168.1.43' - 36 AS RESULT;
result

192.168.1.7
(1 row)
```

- -

Description: Subtraction

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT family('127.0.0.1') AS RESULT;
result
-----
4
(1 row)
```
 - **host(inet)**
Description: Extracts IP addresses as text.
Return type: text
Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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 JSON/JSONB Functions and Operators

For details about the JSON/JSONB data type, see [JSON/JSONB Types](#).

Table 7-33 JSON/JSONB common operators

Operator	Left Oper and Type	Right Oper and Type	Return Type	Description	Example
->	Array - json(b)	int	json(b)	Obtains the array-json element. If the index does not exist, NULL is returned.	<pre>SELECT '["a":"foo"],["b":"bar"], {"c":"baz"}'::json->2; ?column? ----- {"c":"baz"} (1 row)</pre>
->	object - json(b)	text	json(b)	Obtains the value by a key. If no record is found, NULL is returned.	<pre>SELECT '{"a": {"b":"foo"}}'::json->'a'; ?column? ----- {"b":"foo"} (1 row)</pre>

O p e r a t o r	Left Oper and Type	Rig ht Ope ran d Typ e	Re tu rn Ty pe	Description	Example
- > >	Array - json(b)	int	te xt	Obtains the JSON array element. If the index does not exist, NULL is returned.	<pre>SELECT '[1,2,3]::json->>2; ?column? ----- 3 (1 row)</pre>
- > >	objec t- json(b)	text	te xt	Obtains the value by a key. If no record is found, NULL is returned.	<pre>SELECT '{"a":1,"b":2}::json->>'b'; ?column? ----- 2 (1 row)</pre>
# >	conta iner- json (b)	tex t[]	jso n(b)	Obtains the JSON object in the specified path. If the path does not exist, NULL is returned.	<pre>SELECT '{"a":{"b":{"c":"foo"}}}::json #>'a,b'; ?column? ----- {"c": "foo"} (1 row)</pre>
# > >	conta iner- json (b)	tex t[]	te xt	Obtains the JSON object in the specified path. If the path does not exist, NULL is returned.	<pre>SELECT '{"a":[1,2,3],"b":[4,5,6]}::json #>>'a,2'; ?column? ----- 3 (1 row)</pre>

 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 7-34 Additional JSONB support for operators

Operator	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 JSONB files, which is the same as the jsonb_eq function.	/
<>	jsonb	Determines the size between two JSONB files, which is the same as the jsonb_ne function.	/

Operator	Right Operand Type	Description	Example
<	jsonb	Determines the size between two JSONB files, which is the same as the jsonb_lt function.	/
>	jsonb	Determines the size between two JSONB files, which is the same as the jsonb_gt function.	/
<=	jsonb	Determines the size between two JSONB files, which is the same as the jsonb_le function.	/
>=	jsonb	Determines the size between two JSONB files, which is the same as the jsonb_ge function.	/

Functions Supported by JSON/JSONB

- array_to_json(anyarray [, pretty_bool])

Description: Returns an 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

For example:

```
openGauss=# SELECT array_to_json('{{1,5},{99,100}}':int[]);
 array_to_json
-----
 [[1,5],[99,100]]
 (1 row)
```

- row_to_json(record [, pretty_bool])

Description: Returns a row as JSON. Line feeds will be added between level-1 elements if **pretty_bool** is **true**.

Return type: json

For example:

```
openGauss=# SELECT row_to_json(row(1,'foo'));
 row_to_json
-----
 [{"1":1,"foo":"foo"}]
```

```
-----  
{ "f1":1,"f2":"foo"} (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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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: Same as the operator ``#>`` and searches for JSON based on the path specified by `$2` and returns the result.

Return type: json, jsonb

For example:

```
openGauss=# 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: It is equivalent to the operator ``#>``, searches for JSON based on the path specified by `$2`, and returns the result.

Return type: json, jsonb

For example:

```
openGauss=# 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: It is equivalent to the operator ``#>>``, searches for JSON based on the path specified by `$2`, and returns the result.

Return type: text, text

For example:

```
openGauss=# 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: It is equivalent to the operator ``#>>>``, searches for JSON based on the path specified by `$2`, and returns the result.

Return type: text, text

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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)`

For example:

```
openGauss=# 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)`

For example:

```
openGauss=# 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

For example:

```
openGauss=# SELECT json_object_keys('{\"f1\":\"abc\", \"f2\":{\"f3\":\"a\", \"f4\":\"b\"}, \"f1\":\"abcd\"}');
 json_object_keys
-----
 f1
 f2
 f1
(3 rows)
```

- JSONB deduplication operations:

```
openGauss=# SELECT jsonb_object_keys('{\"f1\":\"abc\", \"f2\":{\"f3\":\"a\", \"f4\":\"b\"}, \"f1\":\"abcd\"}');
 jsonb_object_keys
-----
```

```
f1  
f2  
(2 rows)
```

- `json_populate_record(anyelement, object-json [, bool])`,
`jsonb_populate_record(anyelement, object-jsonb [, bool])`

Description: *\$1* must be a compound parameter. Each key-value in the **object-json** file is split. The key is used as the column name to match the column name in *\$1* and fill in the *\$1* format.

Return type: anyelement, anyelement

For example:

```
openGauss=# CREATE TYPE jpop as (a text, b int, c bool);  
CREATE TYPE  
openGauss=# SELECT * FROM json_populate_record(null::jpop, '{"a":"blurfl","x":43.2}');  
 a | b | c  
-----+-----+---  
blurfl | |  
(1 row)
```

```
openGauss=# SELECT * FROM json_populate_record((1,1,null)::jpop, '{"a":"blurfl","x":43.2}');  
 a | b | c  
-----+-----+---  
blurfl | 1 |  
(1 row)
```

- `json_populate_record_set(anyelement, array-json [, bool])`,
`jsonb_populate_record_set(anyelement, array-jsonb [, bool])`

Description: Performs the preceding operations on each element in the *\$2* array by referring to the **json_populate_record** and **jsonb_populate_record** functions. Therefore, each element in the *\$2* array must be of the **object-json** type.

Return type: setof anyelement, setof anyelement

For example:

```
openGauss=# CREATE TYPE jpop as (a text, b int, c bool);  
CREATE TYPE  
openGauss=# SELECT * FROM json_populate_recordset(null::jpop, '[{"a":1,"b":2}, {"a":3,"b":4}]');  
 a | b | c  
---+---+---  
 1 | 2 |  
 3 | 4 |  
(2 rows)
```

- `json_typeof(json)`, `jsonb_typeof(jsonb)`

Description: Checks the JSON type.

Return type: text, text

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# SELECT json_build_object(1,2);
           json_build_object
-----
{"1" : 2}
(1 row)
```

- `json_to_record(object-json, bool)`

Description: Like all functions that return **record**, the caller must explicitly define the structure of the record with an AS clause. The key-value pair of **object-json** is split and reassembled. The key is used as a column name to match and fill in the structure of the specified record.

Return type: record

For example:

```
openGauss=# SELECT * FROM json_to_record('{"a":1,"b":"foo","c":"bar"}',true) as x(a int, b text, d text);
 a | b | d
---+---+---
 1 | foo | 
(1 row)
```

- `json_to_recordset(array-json, bool)`

Description: Executes the preceding function on each element in the array by referring to the `json_to_record` function. Therefore, each element in the array must be object-json.

Return type: SETOF record

For example:

```
openGauss=# SELECT * FROM json_to_recordset(
openGauss(#  '{"a":1,"b":"foo","d":false},{a":2,"b":"bar","c":true}'],
openGauss(#  false
openGauss(# ) as x(a int, b text, c boolean);
 a | b | c
---+---+---
 1 | foo | 
 2 | bar | t
(2 rows)
```

- `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

For example:

```
openGauss=# 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)
openGauss=# 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

For example:

```
openGauss=# SELECT * FROM classes;
 name | score
-----+-----
 A    |    2
 A    |    3
 D    |    5
 D    |
(4 rows)
openGauss=# 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

For example:

```
openGauss=# SELECT * FROM classes;
 name | score
-----+-----
 A    |    2
 A    |    3
 D    |    5
 D    |
(4 rows)
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# 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

For example:

```
openGauss=# SELECT jsonb_hash('[1,2,3]');
jsonb_hash
-----
-559968547
(1 row)
```

- **Other functions**

Description: Internal functions used by JSON and JSONB aggregate functions.

```
json_agg_transfn
json_agg_finalfn
json_object_agg_transfn
json_object_agg_finalfn
```

7.5.13 HLL Functions and Operators

Hash Functions

- **hll_hash_boolean(bool)**

Description: Hashes data of the Boolean type.

Return type: hll_hashval

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT hll_hash_any(1);
hll_hash_any
-----
-1316670585935156930
(1 row)

openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
-----
f
(1 row)
```

Log 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT hll_log2m(hll_empty());
hll_log2m
-----
      14
(1 row)

openGauss=# SELECT hll_log2m(hll_empty(10));
hll_log2m
-----
      10
(1 row)

openGauss=# SELECT hll_log2m(hll_empty(-1));
hll_log2m
-----
      14
(1 row)
```

- **hll_log2explicit(hll)**

Description: Queries the **log2explicit** value 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 explicit mode will be skipped and the sparse mode is directly entered. When the value of **log2explicit** is explicitly set to a value ranging from 1 to 12, HLL will switch to the sparse mode when the length of the data segment exceeds $2^{\log 2explicit}$. When the value of **log2explicit** is explicitly set to **-1**, the built-in default value is used.

Example:

```
openGauss=# SELECT hll_log2explicit(hll_empty());
hll_log2explicit
-----
      10
(1 row)

openGauss=# SELECT hll_log2explicit(hll_empty(12, 8));
hll_log2explicit
-----
       8
(1 row)
```

```
openGauss=# 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, 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:

```
openGauss=# SELECT hll_log2sparse(hll_empty());
hll_log2sparse
-----
          12
(1 row)

openGauss=# SELECT hll_log2sparse(hll_empty(12, 8, 10));
hll_log2sparse
-----
          10
(1 row)

openGauss=# SELECT hll_log2sparse(hll_empty(12, 8, -1));
hll_log2sparse
-----
          12
(1 row)
```

- **hll_duplicatecheck(hll)**

Description: Specifies whether duplicate check is enabled. The value **0** indicates that it is disabled and the value **1** indicates that it is enabled. This function is disabled by default. If there are many duplicate values, you can enable this function to improve efficiency. When the value of **duplicatecheck** is explicitly set to **-1**, the built-in default value is used.

Example:

```
openGauss=# SELECT hll_duplicatecheck(hll_empty());
hll_duplicatecheck
-----
          0
(1 row)

openGauss=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, 1));
hll_duplicatecheck
-----
          1
(1 row)

openGauss=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, -1));
hll_duplicatecheck
-----
          0
(1 row)
```


Example:

```
openGauss=# SELECT hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_eq
-----
f
(1 row)
```

- **hll_ne(hll, hll)**

Description: Compares two HLLs to check whether they are different.

Return type: Boolean

Example:

```
openGauss=# SELECT hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_ne
-----
t
(1 row)
```

- **hll_cardinality(hll)**

Description: Calculates the number of distinct values of an HLL.

Return type: int

Example:

```
openGauss=# SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----
1
(1 row)
```

- **hll_union(hll, hll)**

Description: Performs an UNION operation on two HLL data structures to obtain one HLL.

Return type: HLL

Example:

```
openGauss=# 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.
openGauss=# CREATE TABLE t_id(id int);
openGauss=# INSERT INTO t_id VALUES(generate_series(1,500));
openGauss=# CREATE TABLE t_data(a int, c text);
openGauss=# INSERT INTO t_data SELECT mod(id,2), id FROM t_id;

-- Create a table and specify an HLL column.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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 parameters **log2m**, **log2explicit**, and **log2sparse** 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:

```
openGauss=# 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:

```
openGauss=# 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 an 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:
openGauss=# SELECT #hll_union_agg(c) as cardinality FROM t_a_c_hll;
cardinality
-----
498.496062953313
(1 row)

-- Drop tables.
openGauss=# DROP TABLE t_id;
openGauss=# DROP TABLE t_data;
openGauss=# 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 deprecated 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 **regwidth** variable is not used.

- **hll_expthresh(hll)**

Description: Obtains the **expthresh** value in the current HLL. The **hll_log2explicit(hll)** function is used to replace similar functions.

- **hll_sparseon(hll)**

Description: Specifies whether to enable the sparse mode. Use **hll_log2sparse(hll)** to replace similar functions. The value **0** indicates that the sparse mode is disabled.

Built-in Functions

HyperLogLog (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 7-35](#).

Table 7-35 Built-in Functions

Function	Description
hll_in	Receives hll data in string format.
hll_out	Sends hll data in string format.
hll_recv	Receives hll data in bytea format.
hll_send	Sends hll data in bytea format.
hll_trans_in	Receives hll_trans_type data in string format.
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_tpmmod_in	Receives tpmmod data.
hll_tpmmod_out	Sends tpmmod 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.

Function	Description
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 HLL and **hll_hashval** types to check whether they are the same.

Return type: Boolean

Example:

```
--hll
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column
-----
t
(1 row)

--hll_hashval
openGauss=# SELECT hll_hash_integer(1) = hll_hash_integer(1);
?column?
-----
t
(1 row)
```

- **<> or !=**
Description: Compares the values of HLL and **hll_hashval** types to check whether they are different.

Return type: Boolean

Example:

```
--hll
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
?column?
-----
t
(1 row)

--hll_hashval
openGauss=# 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
openGauss=# SELECT hll_empty() || hll_hash_integer(1);
               ?column?
-----
\x484c4c08000002002b090000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
openGauss=# SELECT hll_hash_integer(1) || hll_empty();
               ?column?
-----
\x484c4c08000002002b090000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
               ?column?
-----
\x484c4c10002000002b090000000000000004000000000000000b3ccc49320cca1ae3e2921ff133fbaed00
(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:

```
openGauss=# SELECT #(hll_empty() || hll_hash_integer(1));
               ?column?
-----
               1
(1 row)
```

7.5.14 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 has been 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 DNs, 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 the 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 clusterDNs. 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 Oracle syntax is supported. Currently, the sequence name cannot contain a dot.):

```
openGauss=# CREATE SEQUENCE seqDemo;

-- Example 1:
openGauss=# SELECT nextval('seqDemo');
 nextval
-----
        1
(1 row)

-- Example 2:
openGauss=# SELECT seqDemo.nextval;
 nextval
-----
        2
(1 row)
openGauss=# DROP SEQUENCE seqDemo;
```

- **curral(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 curral 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 curral function can be called in either of the following ways (In example 2, the Oracle syntax is supported. Currently, the sequence name cannot contain a dot.):

```
openGauss=# CREATE SEQUENCE seq1;
openGauss=# SELECT nextval('seq1');
openGauss=# SET enable_beta_features = true;

-- Example 1:
openGauss=# SELECT curral('seq1');
 curral
-----
        1
(1 row)

-- Example 2:
openGauss=# SELECT seq1.curral seq1;
```

```
seq1
-----
1
(1 row)
openGauss=# DROP SEQUENCE seq1;
openGauss=# SET enable_beta_features = false;
```

- **lastval()**

Description: Returns the last value of **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, calling **lastval** will report an error.

By default, **lastval** is disabled. To enable it, set **enable_beta_features** or **lastval_supported** to **true**. After **lastval** is enabled, **nextval** will not be pushed down.

Return type: numeric

Example:

```
openGauss=# CREATE SEQUENCE seq1;
openGauss=# SELECT nextval('seq1');
openGauss=# SET enable_beta_features = true;
openGauss=# SELECT lastval();
lastval
-----
1
(1 row)
openGauss=# DROP SEQUENCE seq1;
openGauss=# SET enable_beta_features = false;
```

- **setval(regclass, bigint)**

Sets the current value of a sequence.

Return type: numeric

Example:

```
openGauss=# CREATE SEQUENCE seqDemo;
openGauss=# SELECT nextval('seqDemo');
openGauss=# SELECT setval('seqDemo',3);
setval
-----
3
(1 row)
openGauss=# DROP SEQUENCE seqDemo;
```

- **setval(regclass, numeric, Boolean)**

Sets the current value of a sequence and the **is_called** sign.

Return type: numeric

Example:

```
openGauss=# CREATE SEQUENCE seqDemo;
openGauss=# SELECT nextval('seqDemo');
openGauss=# SELECT setval('seqDemo',5,true);
setval
-----
5
(1 row)
openGauss=# 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 use **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

7.5.15 Array Functions and Operators

Array Operators

- `=`

Description: Specifies whether two arrays are equal.

Example:

```
openGauss=# 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:

```
openGauss=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
result
-----
t
(1 row)
```

- `<`

Description: Specifies whether an array is less than another.

Example:

```
openGauss=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
result
-----
t
(1 row)
```

- `>`

Description: Specifies whether an array is greater than another.

Example:

```
openGauss=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
result
-----
t
(1 row)
```

- `<=`

Description: Specifies whether an array is less than or equal to another.

Example:

```
openGauss=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
result
-----
t
(1 row)
```

- >=

Description: Specifies whether an array is greater than or equal to another.

Example:

```
openGauss=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
result
-----
t
(1 row)
```

- @>

Description: Specifies whether an array contains another.

Example:

```
openGauss=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
result
-----
t
(1 row)
```

- <@

Description: Specifies whether an array is contained in another.

Example:

```
openGauss=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
result
-----
t
(1 row)
```

- &&

Description: Specifies whether an array overlaps another (have common elements).

Example:

```
openGauss=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
result
-----
t
(1 row)
```

- ||

Description: Specifies array-to-array concatenation.

Example:

```
openGauss=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
result
-----
{1,2,3,4,5,6}
(1 row)
openGauss=# 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: Specifies element-to-array concatenation.

Example:

```
openGauss=# SELECT 3 || ARRAY[4,5,6] AS RESULT;  
result  
-----  
{3,4,5,6}  
(1 row)
```

- ||

Description: Specifies array-to-element concatenation.

Example:

```
openGauss=# 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 number of dimensions 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. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# 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. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# 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:

```
openGauss=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;  
result  
-----  
{1,2,3,4,5}  
(1 row)  
  
openGauss=# 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.

Return type: anyarray

Example:

```
openGauss=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
```

```
result
-----
{1,2,3,3,4,5}
(1 row)
```

- `array_union_distinct(anyarray, anyarray)`

Description: Concatenates two arrays and deduplicates them. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
```

```
result
-----
{1,2,3,4,5}
(1 row)
```

- `array_intersect(anyarray, anyarray)`

Description: Intersects two arrays. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
```

```
result
-----
{3}
(1 row)
```

- `array_intersect_distinct(anyarray, anyarray)`

Description: Intersects two arrays and deduplicates them. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
```

```
result
-----
{2}
(1 row)
```

- `array_except(anyarray, anyarray)`

Description: Calculates the difference between two arrays. Only one-dimensional arrays are supported.

Return type: anyarray

Example:

```
openGauss=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
```

```
result
-----
{1,2}
(1 row)
```

- **array_except_distinct(anyarray, anyarray)**
Description: Calculates the difference between two arrays and deduplicates them. Only one-dimensional arrays are supported.
Return type: anyarray
Example:

```
openGauss=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
result
-----
{1,2}
(1 row)
```
- **array_ndims(anyarray)**
Description: Returns the number of dimensions of an array.
Return type: int
Example:

```
openGauss=# 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 dimensions.
Return type: text
Example:

```
openGauss=# 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 array dimensions.
Return type: int
Example:

```
openGauss=# SELECT array_length(array[1,2,3], 1) AS RESULT;
result
-----
3
(1 row)
```
- **array_lower(anyarray, int)**
Description: Returns lower bound of the array dimensions.
Return type: int
Example:

```
openGauss=# 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:

```
openGauss=# 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 array dimensions.

Return type: int

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;  
result  
-----  
{2,3,4,5}  
(1 row)
```

- `array_extendnull(anyarray, int)`

Description: Adds a specified number of null elements to the end of an array.

Return type: anyarray

Example:

```
openGauss=# SELECT array_extend(ARRAY[1,8,3,7],1) AS RESULT;  
result  
-----  
{1,8,3,7,null}  
(1 row)
```


- `array_trim(anyarray, int)`

Description: Deletes a specified number of elements from the end of an array.

Return type: `anyarray`

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
result
-----
{xx,NULL,zz}
(1 row)
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
result
-----
```

```
{xx,yy,zz}  
(1 row)
```

- **unnest(anyarray)**

Description: Expands an array to a set of rows.

Return type: setof anyelement

Example:

```
openGauss=# 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.

7.5.16 Range Functions and Operators

Range Operators

- **=**

Description: Equals

Example:

```
openGauss=# SELECT int4range(1,5) = '[1,4]::int4range AS RESULT;  
result  
-----  
t  
(1 row)
```

- **<>**

Description: Does not equal to

Example:

```
openGauss=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;  
result  
-----  
t  
(1 row)
```

- **<**

Description: Is less than

Example:

```
openGauss=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
result
-----
t
(1 row)
```

- >

Description: Is greater than

Example:

```
openGauss=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
result
-----
t
(1 row)
```

- <=

Description: Is less than or equals

Example:

```
openGauss=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
result
-----
t
(1 row)
```

- >=

Description: Is greater than or equals

Example:

```
openGauss=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
result
-----
t
(1 row)
```

- @>

Description: Contains ranges

Example:

```
openGauss=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
result
-----
t
(1 row)
```

- @>

Description: Contains elements

Example:

```
openGauss=# SELECT '[2011-01-01,2011-03-01) '::tsrange @> '2011-01-10 '::timestamp AS RESULT;
result
-----
t
(1 row)
```

- <@

Description: Range is contained by

Example:

```
openGauss=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
result
-----
t
(1 row)
```

- <@

Description: Element is contained by

Example:

```
openGauss=# SELECT 42 <@ int4range(1,7) AS RESULT;
result
-----
f
(1 row)
```
- &&

Description: Overlap (have points in common)

Example:

```
openGauss=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
result
-----
t
(1 row)
```
- <<

Description: Strictly left of

Example:

```
openGauss=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
result
-----
t
(1 row)
```
- >>

Description: Strictly right of

Example:

```
openGauss=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
result
-----
t
(1 row)
```
- &<

Description: Does not extend to the right of

Example:

```
openGauss=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
result
-----
t
(1 row)
```
- &>

Description: Does not extend to the left of

Example:

```
openGauss=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
result
-----
t
(1 row)
```
- -|-

Description: Is adjacent to

Example:

```
openGauss=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
result
```

- t
(1 row)
- +
Description: Union
Example:
openGauss=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
result

[5,20)
(1 row)
- *
Description: Intersection
Example:
openGauss=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
result

[10,15)
(1 row)
- -
Description: Difference
Example:
openGauss=# 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:
openGauss=# SELECT numrange(1.1,2.2) AS RESULT;
result

[1.1,2.2)
(1 row)
openGauss=# 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:

```
openGauss=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
result
-----
1.1
(1 row)
```

- **upper(anyrange)**

Description: Upper bound of a range

Return type: Range's element type

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT lower_inf('(',')::daterange) AS RESULT;
result
-----
t
(1 row)
```

- `upper_inf(anyrange)`

Description: Is the upper bound infinite?

Return type: Boolean

Example:

```
openGauss=# SELECT upper_inf('(',')::daterange) AS RESULT;
result
-----
t
(1 row)
```

- `elem_contained_by_range(anyelement, anyrange)`

Description: Determines whether an element is within the range.

Return type: Boolean

Example:

```
openGauss=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
elem_contained_by_range
-----
t
(1 row)
```

7.5.17 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:

```
openGauss=# CREATE TABLE tab(a int);
CREATE TABLE
openGauss=# INSERT INTO tab values(1);
INSERT 0 1
openGauss=# INSERT INTO tab values(2);
INSERT 0 1
openGauss=# 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:

```
openGauss=# CREATE TABLE max_t1(a int, b int);

openGauss=# INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT MAX(a) FROM max_t1;
max
-----
4
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TABLE min_t1(a int, b int);

openGauss=# INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT MIN(a) FROM min_t1;
min
-----
1
(1 row)

openGauss=# 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 arguments.

Otherwise, it is the same as the input data type.

Example:

```
openGauss=# CREATE TABLE avg_t1(a int, b int);

openGauss=# INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT AVG(a) FROM avg_t1;
avg
-----
2.5000000000000000
(1 row)

openGauss=# 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

Example:

```
openGauss=# CREATE TABLE count_t1(a int, b int);

openGauss=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);
```



```
openGauss=# SELECT COUNT(a) FROM count_t1;
count
-----
      4
(1 row)

openGauss=# DROP TABLE count_t1;
```

- **count(*)**

Description: Returns the number of input rows.

Return type: bigint

Example:

```
openGauss=# CREATE TABLE count_t1(a int, b int);

openGauss=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT COUNT(*) FROM count_t1;
count
-----
      5
(1 row)

openGauss=# DROP TABLE count_t1;
```

- **array_agg(expression)**

Description: Concatenates input values, including nulls, into an array.

Return type: array of the argument type

Example:

```
openGauss=# CREATE TABLE array_agg_t1(a int, b int);

openGauss=# INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT ARRAY_AGG(a) FROM array_agg_t1;
array_agg
-----
{NULL,1,2,3,4}
(1 row)

openGauss=# 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

Example:

```
openGauss=# CREATE TABLE string_agg_t1(a int, b int);

openGauss=# INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT STRING_AGG(a,',') FROM string_agg_t1;
string_agg
-----
1;2;3;4
(1 row)

openGauss=# 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.

```
openGauss=# CREATE TABLE listagg_t1(a int, b text);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

openGauss=# 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)

openGauss=# DROP TABLE listagg_t1;
```

The aggregation column is of the integer type.

```
openGauss=# CREATE TABLE listagg_t1(a int, b int);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE listagg_t1;
```

The aggregation column is of the floating point type.

```
openGauss=# CREATE TABLE listagg_t1(a int, b float);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),(2,5.555),
(3,6.666);

openGauss=# 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)
```

```
openGauss=# DROP TABLE listagg_t1;
```

The aggregation column is of the time type.

```
openGauss=# CREATE TABLE listagg_t1(a int, b timestamp);
```

```
openGauss=# 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');
```

```
openGauss=# 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)
```

```
openGauss=# DROP TABLE listagg_t1;
```

The aggregation column is of the time interval type.

```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);
```

```
openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5  
days'),(3,'6 days');
```

```
openGauss=# 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)
```

```
openGauss=# DROP TABLE listagg_t1;
```

By default, the delimiter is empty.

```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);
```

```
openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5  
days'),(3,'6 days');
```

```
openGauss=# 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)
```

```
openGauss=# 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.

```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);
```

```
openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5  
days'),(3,'6 days');
```

```
openGauss=# 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)
```

```
openGauss=# DROP TABLE listagg_t1;
```

- **covar_pop(Y, X)**

Description: Specifies the overall covariance.

Return type: double precision

Example:

```
openGauss=# CREATE TABLE covar_pop_t1(a int, b int);
```

```
openGauss=# INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

```
openGauss=# SELECT COVAR_POP(a,b) FROM covar_pop_t1;
```

```
 covar_pop
```

```
-----
      100
(1 row)
```

```
openGauss=# DROP TABLE covar_pop_t1;
```

- **covar_samp(Y, X)**

Description: Specifies the sample covariance.

Return type: double precision

Example:

```
openGauss=# CREATE TABLE covar_samp_t1(a int, b int);
```

```
openGauss=# INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

```
openGauss=# SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
```

```
 covar_samp
```

```
-----
      125
(1 row)
```

```
openGauss=# 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:

```
openGauss=# CREATE TABLE stddev_pop_t1(a int, b int);
```

```
openGauss=# INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```

```
openGauss=# SELECT STDDEV_POP(a) FROM stddev_pop_t1;
```

```
 stddev_pop
```

```
-----
7.4833147735478828
(1 row)
```

```
openGauss=# 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:

```
openGauss=# CREATE TABLE stddev_samp_t1(a int, b int);
openGauss=# INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
openGauss=# SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
      stddev_samp
-----
8.3666002653407555
(1 row)
openGauss=# 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:

```
openGauss=# CREATE TABLE var_pop_t1(a int, b int);
openGauss=# INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
openGauss=# SELECT VAR_POP(a) FROM var_pop_t1;
      var_pop
-----
56.0000000000000000
(1 row)
openGauss=# 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:

```
openGauss=# CREATE TABLE var_samp_t1(a int, b int);
openGauss=# INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
openGauss=# SELECT VAR_SAMP(a) FROM var_samp_t1;
      var_samp
-----
70.0000000000000000
(1 row)
openGauss=# 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:

```
openGauss=# CREATE TABLE bit_and_t1(a int, b int);
openGauss=# INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
openGauss=# SELECT BIT_AND(a) FROM bit_and_t1;
      bit_and
-----
0
(1 row)
```

```
openGauss=# 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:

```
openGauss=# CREATE TABLE bit_or_t1(a int, b int);
```

```
openGauss=# INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
openGauss=# SELECT BIT_OR(a) FROM bit_or_t1;
```

```
bit_or
```

```
-----
```

```
3
```

```
(1 row)
```

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# SELECT bool_or(100 < 2500);
```

```
bool_or
```

```
-----
```

```
t
```

```
(1 row)
```

- **corr(Y, X)**

Description: Specifies the correlation coefficient.

Return type: double precision

Example:

```
openGauss=# CREATE TABLE corr_t1(a int, b int);
```

```
openGauss=# INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
openGauss=# SELECT CORR(a,b) FROM corr_t1;
```

```
corr
```

```
-----
```

```
.944911182523068
```

```
(1 row)
```

```
openGauss=# DROP TABLE corr_t1;
```

- **every(expression)**

Description: Equivalent to **bool_and**

Return type: Boolean

Example:

```
openGauss=# 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:

```
openGauss=# CREATE TABLE rank_t1(a int, b int);

openGauss=# INSERT INTO rank_t1 VALUES(NULL,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE rank_t1;
```

- **regr_avgx(Y, X)**

Description: Specifies the average of the independent variable (**sum(X)/Y**).

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_t1(a int, b int);

openGauss=# INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_AVGX(a,b) FROM regr_t1;
regr_avgx
-----
4
(1 row)

openGauss=# DROP TABLE regr_t1;
```

- **regr_avgy(Y, X)**

Description: Specifies the average of the dependent variable (**sum(Y)/X**).

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_avgy_t1(a int, b int);

openGauss=# INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
regr_avgy
-----
1.8
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TABLE regr_count_t1(a int, b int);

openGauss=# INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_COUNT(a,b) FROM regr_count_t1;
 regr_count
-----
          5
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TABLE regr_intercept_t1(a int, b int);

openGauss=# INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
 regr_intercept
-----
.785714285714286
(1 row)

openGauss=# DROP TABLE regr_intercept_t1;
```

- **regr_r2(Y, X)**

Description: Specifies the square of the correlation coefficient.

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_r2_t1(a int, b int);

openGauss=# INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_R2(b,a) FROM regr_r2_t1;
 regr_r2
-----
.892857142857143
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TABLE regr_slope_t1(a int, b int);

openGauss=# INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
 regr_slope
```



```
-----  
1.78571428571429  
(1 row)
```

```
openGauss=# DROP TABLE regr_slope_t1;
```

- **regr_sxx(Y, X)**

Description: **$\text{sum}(Y^2) - \text{sum}(X)^2/N$** (sum of squares of the independent variables)

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_sxx_t1(a int, b int);
```

```
openGauss=# INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
openGauss=# SELECT REGR_SXX(b,a) FROM regr_sxx_t1;  
regr_sxx
```

```
-----  
2.8  
(1 row)
```

```
openGauss=# DROP TABLE regr_sxx_t1;
```

- **regr_sxy(Y, X)**

Description: **$\text{sum}(X*Y) - \text{sum}(X) * \text{sum}(Y)/N$** ("sum of products" of independent times dependent variable)

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_sxy_t1(a int, b int);
```

```
openGauss=# INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
openGauss=# SELECT REGR_SXY(b,a) FROM regr_sxy_t1;  
regr_sxy
```

```
-----  
5  
(1 row)
```

```
openGauss=# DROP TABLE regr_sxy_t1;
```

- **regr_syy(Y, X)**

Description: **$\text{sum}(Y^2) - \text{sum}(X)^2/N$** ("sum of squares" of the dependent variable)

Return type: double precision

Example:

```
openGauss=# CREATE TABLE regr_syy_t1(a int, b int);
```

```
openGauss=# INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```

```
openGauss=# SELECT REGR_SYY(b,a) FROM regr_syy_t1;  
regr_syy
```

```
-----  
10  
(1 row)
```

```
openGauss=# 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:

```
openGauss=# CREATE TABLE stddev_t1(a int, b int);

openGauss=# INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT STDDEV(a) FROM stddev_t1;
          stddev
-----
.83666002653407554798
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TABLE variance_t1(a int, b int);

openGauss=# INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT VARIANCE(a) FROM variance_t1;
          variance
-----
.70000000000000000000
(1 row)

openGauss=# 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

- **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 the **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 TEXT types 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 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:

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(a) FROM checksum_t1;
checksum
-----
18126842830
(1 row)

openGauss=# 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*.

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
checksum
-----
18126842830
(1 row)

openGauss=# 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.

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
checksum
-----
11160522226
(1 row)

openGauss=# DROP TABLE checksum_t1;
```

7.5.18 Window Functions

Window Functions

Window functions and the **OVER** clause are used together. 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: Generates non-consecutive sequence numbers for the values in each group. The same values have the same sequence number.

Return type: bigint

Example:

```
openGauss=# CREATE TABLE rank_t1(a int, b int);

openGauss=# INSERT INTO rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE rank_t1;
```

- **ROW_NUMBER()**

Description: Generates consecutive sequence numbers for the values in each group. The same values have different sequence numbers.

Return type: bigint

Example:

```
openGauss=# CREATE TABLE row_number_t1(a int, b int);

openGauss=# INSERT INTO row_number_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE row_number_t1;
```

- **DENSE_RANK()**

Description: Generates consecutive sequence numbers for the values in each group. The same values have the same sequence number.

Return type: bigint

Example:

```
openGauss=# CREATE TABLE dense_rank_t1(a int, b int);

openGauss=# INSERT INTO dense_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# 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 = $(\text{rank} - 1) / (\text{totalrows} - 1)$. **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:

Example:

```
openGauss=# CREATE TABLE percent_rank_t1(a int, b int);

openGauss=# INSERT INTO percent_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE percent_rank_t1;
```

- CUME_DIST()

Description: Generates 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:

```
openGauss=# CREATE TABLE cume_dist_t1(a int, b int);
```

```
openGauss=# INSERT INTO cume_dist_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE cume_dist_t1;
```

- NTILE(num_buckets integer)

Description: Equally allocates sequential data sets to the buckets whose quantity is specified by **num_buckets** according to **num_buckets integer** and allocates the bucket number to each row. Divide the partition as evenly as possible.

Return type: integer

Example:

```
openGauss=# CREATE TABLE ntile_t1(a int, b int);

openGauss=# INSERT INTO ntile_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# DROP TABLE ntile_t1;
```

- LAG(value any [, offset integer [, default any]])

Description: Generates 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.
openGauss=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
openGauss=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-JUL-05', 'Tobias', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-DEC-05', 'Baida', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('15-NOV-06', 'Himuro', 30);
INSERT 0 1
```

```
openGauss=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-07', 'yq', 11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-08', 'zi', 11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('', 'yq1', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, 'yq2', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-DEC-07', 'yq3', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1

-- Call LAG and set offset to 3 and default to null.
openGauss=# 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)

openGauss=# DROP TABLE ta1;
```

- LEAD(value any [, offset integer [, default any]])

Description: Generates 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:

```
openGauss=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
openGauss=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-JUL-05', 'Tobias', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-DEC-05', 'Baida', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('15-NOV-06', 'Himuro', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-07', 'yq', 11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-08', 'zi', 11);
```

```

INSERT 0 1
openGauss=# INSERT INTO ta1 values('yq1', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, 'yq2', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-DEC-07', 'yq3', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null, 11);
INSERT 0 1

-- Call LEAD and set offset to 2.
openGauss=# 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)

openGauss=# DROP TABLE ta1;

```

- **FIRST_VALUE(value any)**

Description: Returns the first value of each group.

Return type: same as the parameter type

Example:

```

openGauss=# CREATE TABLE first_value_t1(a int, b int);

openGauss=# INSERT INTO first_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

openGauss=# 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:

```

openGauss=# CREATE TABLE last_value_t1(a int, b int);

openGauss=# INSERT INTO last_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# 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)

```
openGauss=# DROP TABLE last_value_t1;
```

- **NTH_VALUE(value any, nth integer)**

Description: Returns the *n*th row for a group. If the row does not exist, **NULL** is returned by default.

Return type: same as the parameter type

Example:

```
openGauss=# CREATE TABLE nth_value_t1(a int, b int);
```

```
openGauss=# INSERT INTO nth_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);
```

```
openGauss=# 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	
2	5	5
3	6	

(7 rows)

```
openGauss=# 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

7.5.19 Security Functions

Security Functions

- **gs_encrypt_aes128(encryptstr,keyststr)**

Description: Encrypts **encryptstr** strings using the key derived from **keyststr** and returns encrypted strings. The value of **keyststr** 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:

```
openGauss=# SELECT gs_encrypt_aes128('MPPDB','1234@abc');
               gs_encrypt_aes128
-----
OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Ttt37fZF8Q+bbEYhdw/YG+0c9tHKRWM6OcTzIB3HnqvX
+1d8Bflo=
(1 row)
```

NOTE

A password is required during the execution of this function. For security purposes, the gsqr 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 gsqr by paging up and down.

- `gs_decrypt_aes128(decryptstr,keyststr)`

Description: Decrypts **decrypt** strings using the key derived from **keyststr** and returns decrypted strings. The **keyststr** used for decryption must be consistent with that used for encryption. **keyststr** cannot be empty.

NOTE

This function needs to be used with the **gs_encrypt_aes128** encryption function.

Return type: text

Example:

```
openGauss=# SELECT gs_decrypt_aes128('OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Ttt37fZF8Q
+bbEYhdw/YG+0c9tHKRWM6OcTzIB3HnqvX+1d8Bflo=', '1234@abc');
               gs_decrypt_aes128
-----
MPPDB
(1 row)
```

NOTE

A password is required during the execution of this function. For security purposes, the gsqr 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 gsqr by paging up and down.

- `gs_password_deadline`

Description: Indicates the number of remaining days before the password of the current user expires.

Return type: interval

Example:

```
openGauss=# 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.

```
openGauss=> 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    | gsqr@[local]
(1 row)
```

- Check the number, date, and time of failed attempts since the previous successful login.

```
openGauss=> 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.

```
openGauss=> 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.

```
openGauss=> 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:

```
openGauss=# 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:

```
openGauss=# 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

Name	Type	Description
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,keyst, encrypttype)**

Description: Encrypts **encryptstr** strings using **keyst** as the key and returns encrypted strings based on **encrypttype**. The value of **keyst** 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:

```
openGauss=# SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
gs_encrypt
-----
ZBzOmaGA4Bb+coyucJ0B8AkIShq
(1 row)
```

NOTE

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.

- **gs_decrypt(decryptstr,keyst,decrypttype)**

Description: Decrypts **decrypt** strings using **keyst** as the key and returns decrypted strings based on **decrypttype**. The **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:

```
openGauss=# SELECT gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShq','Asdf1234','sm4');
gs_decrypt
-----
MPPDB
(1 row)
```

NOTE

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.

7.5.20 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:

```
openGauss=# SELECT * FROM generate_series(2,4);
generate_series
-----
          2
          3
          4
(3 rows)

openGauss=# SELECT * FROM generate_series(5,1,-2);
generate_series
-----
          5
          3
          1
(3 rows)

openGauss=# SELECT * FROM generate_series(4,3);
generate_series
-----
(0 rows)

-- This example applies to the date-plus-integer operator.
openGauss=# 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)

openGauss=# 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)
```

Index Generating Functions

- `generate_subscripts(array anyarray, dim int)`
Description: Generates a series comprising the given array's indexes.
Return type: setof int
- `generate_subscripts(array anyarray, dim int, reverse boolean)`
Description: Generates a series comprising the given array's indexes. 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 indexes 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 indexes are returned for NULL array elements). Example:

```
-- Basic usage
openGauss=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
-- Unnest a 2D array:
openGauss=# 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;

openGauss=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
unnest2
-----
 1
 2
 3
 4
(4 rows)

-- Delete the function.
openGauss=# DROP FUNCTION unnest2;
```


7.5.21 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:

```
openGauss=# 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 not-**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 **base_expr** does not match each **compare(n)**, the default value is returned.

Example:

```
openGauss=# SELECT decode('A','A',1,'B',2,0);
case
-----
1
(1 row)
```

- `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:

```
openGauss=# SELECT nullif('hello','world');
nullif
-----
hello
(1 row)
```

Note:

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 returned. Example:

```
openGauss=# SELECT nullif('1234'::VARCHAR,123::INT4);
nullif
-----
```

```
1234
(1 row)
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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.

- **greatest(expr1 [, ...])**

Description: Selects the largest value from a list of any number of expressions.

Return type:

Example:

```
openGauss=# SELECT greatest(1*2,2-3,4-1);
greatest
-----
3
(1 row)
openGauss=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
greatest
-----
HARRY
(1 row)
```

- **least(expr1 [, ...])**

Description: Selects the smallest value from a list of any number of expressions.

Example:

```
openGauss=# SELECT least(1*2,2-3,4-1);
least
-----
-1
(1 row)
openGauss=# SELECT least('HARRY','HARRIOT','HAROLD');
least
-----
HAROLD
(1 row)
```

- **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.
openGauss=# CREATE TABLE blob_tb(b blob,id int) DISTRIBUTE BY REPLICATION;
-- Insert data.
openGauss=# INSERT INTO blob_tb VALUES (empty_blob(),1);
-- Drop the table.
openGauss=# DROP TABLE blob_tb;
```

Note: The length is 0 obtained using **DBE_LOB.GET_LENGTH**.

7.5.22 System Information Functions

Session Information Functions

- **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:

```
openGauss=# SELECT current_query();
current_query
-----
SELECT current_query();
(1 row)
```

- **current_schema[()]**

Description: Name of the current schema

Return type: name

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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 in the current operating environment.

Return type: text

Example:

```
openGauss=# 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 disabled, the actual session ID is the thread ID.

- `pg_current_sessid`

Description: Session ID in the current operating environment.

Return type: text

Example:

```
openGauss=# select pg_current_sessid();
pg_current_sessid
-----
140308875015936
(1 row)
```

Note: In thread pool mode, the 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:

```
openGauss=# SELECT pg_current_userid();
pg_current_userid
-----
10
(1 row)
```

- `tablespace_oid_name(oid)`

Description: Queries the tablespace name based on the tablespace OID.

Return type: text

Example:

```
openGauss=# 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

This function is valid only in remote connection mode.

Return type: inet

Example:

```
openGauss=# 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

This function is valid only in remote connection mode.

Return type: int

Example:

```
openGauss=# 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

This function is valid only in remote connection mode.

Return type: inet

Example:

```
openGauss=# 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

This function is valid only in remote connection mode.

Return type: int

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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 on.
Return type: setof text
Example:

```
openGauss=# 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 on.
- **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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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.

```
openGauss=# 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:

```
openGauss=# select gs_deployment();
gs_deployment
-----
Distribute
(1 row)
```

- session_user

Description: Session username.

Return type: name

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# select get_shard_oids_byname('datanode1');
get_shard_oids_byname
-----
{16385}
(1 row)
```

- getpgusername()

Description: Obtains the database username.

Return type: name

Example:

```
openGauss=# select getpgusername();
getpgusername
-----
GaussDB_userna
(1 row)
```

- getdatabaseencoding()

Description: Obtains the database encoding mode.

Return type: name

Example:

```
openGauss=# select getdatabaseencoding();
getdatabaseencoding
```

```
-----  
SQL_ASCII  
(1 row)
```

- `version()`

Description: Version information. **version** returns a string describing a server's version.

Return type: text

Example:

```
openGauss=# SELECT version();
```

```
version  
-----  
openGauss 2.0.0 (GaussDB VxxxRxxxCxx build f521c606) compiled at 2021-09-16 14:55:22 commit  
2935 last mr 6385 release  
(1 row)
```

- `working_version_num()`

Description: Returns a version number regarding system compatibility.

Return type: int

Example:

```
openGauss=# 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:

```
openGauss=# SELECT get_hostname();  
get_hostname
```

```
-----  
linux-user  
(1 row)
```

- `get_nodename()`

Description: Returns the name of the current node.

Return type: text

Example:

```
openGauss=# SELECT get_nodename();  
get_nodename
```

```
-----  
coordinator1  
(1 row)
```

- `get_schema_oid(cstring)`

Description: Returns the OID of the queried schema.

Return type: oid

Example:

```
openGauss=# 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: set of record

Example:

```
openGauss=# 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:

```
openGauss=# 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: set of record

Example:

```
openGauss=# 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:

```
openGauss=# 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 7-36 Return value description

Name	Type	Description
nodename	text	Node name
memorytype	text	Memory type. The value must be one of the following: • max_process_memory: memory occupied by a GaussDB cluster instance • process_used_memory: memory occupied by a GaussDB process • max_dynamic_memory: maximum dynamic memory • dynamic_used_memory: used dynamic memory • dynamic_peak_memory: dynamic peak memory • dynamic_used_shrctx: maximum dynamic shared memory context • dynamic_peak_shrctx: dynamic peak value of the shared memory context • max_shared_memory: maximum shared memory • shared_used_memory: used shared memory • max_sctpcomm_memory: maximum memory allowed for the communications library • sctpcomm_used_memory: memory used by the communications library • sctpcomm_peak_memory: memory peak of the communications library • other_used_memory: other used memory
memorybytes	integer	Size of allocated 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 function only specifies whether a user has a certain permission on an object. That is, the permission on the object recorded in the **acl** column of the 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 7-37 Parameter types

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
table	text, oid	Table	Table name or ID
privilege	text	Permission	• SELECT: allows the SELECT statement to be executed on any column of a specified table. • INSERT: allows the INSERT statement to be executed on any column of a specified table. • UPDATE: allows the UPDATE statement to be executed on any column of a specified table. • REFERENCES: allows users to create a foreign key constraint (not supported in distributed mode). • COMMENT: allows the COMMENT statement to be executed on any column of a specified table.

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 7-37](#).

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 7-38 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	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	• SELECT: allows the SELECT statement to be executed on specified columns of a table. • INSERT: allows the INSERT statement to be executed on specified columns of a table. • UPDATE: allows the UPDATE statement to be executed on specified columns of a table. • REFERENCES: allows users to create a foreign key constraint (not supported in distributed mode). • COMMENT: allows the COMMENT statement to be executed on specified columns of a table.

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 7-38](#).

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_database_privilege**(user, database, privilege)
Description: Specifies whether a specified user has permissions on databases.

Table 7-39 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
database	text, oid	Database	Database name or ID
privilege	text	Permission	• CREATE: For databases, allows new schemas to be created within the database. • TEMPORARY: allows users to create temporary tables when the database is used. • TEMP: allows users to create temporary tables when the database is used. • CONNECT: allows users to access specified databases. • ALTER: allows users to modify the attributes of a specified object. • DROP: allows users to delete specified objects. • COMMENT: allows users to define or modify comments of a specified object.

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 7-39](#).

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 7-40 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
directory	text, oid	Directory	Directory name or OID
privilege	text	Permission	• READ: allows read operations on the directory. • WRITE: allows write operations on the directory.

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 7-40](#).

Return type: Boolean

- `has_foreign_data_wrapper_privilege(user, fdw, privilege)`

Table 7-41 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
fdw	text, oid	Foreign data wrapper	Name or ID of the foreign data wrapper
privilege	text	Permission	USAGE : 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 7-41](#).

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 7-42 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
function	text, oid	Function	Function name or ID
privilege	text	Permission	EXECUTE : allows users to use specified functions and the operators that are realized by the functions. • ALTER: allows users to modify the attributes of a specified object. • DROP: allows users to delete a specified object. • COMMENT: allows users to define or modify comments of a specified object.

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 7-42](#).

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 7-43 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Username or ID
language	text, oid	Language	Language name or ID
privilege	text	Permission	USAGE: 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 7-43](#).

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 the permission to access a cluster node.

Return type: Boolean

Table 7-44 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Existing username or ID
nodegroup	text, oid	Cluster node	Existing cluster node

Parameter	Valid Input Parameter Type	Description	Value Range
privilege	text	Permission	• USAGE: For sub-clusters, allows users who can access objects contained in the schema to access tables in the sub-cluster. • CREATE: For sub-clusters, allows users to create tables within the sub-cluster. • COMPUTE: allows users to perform elastic computing in the sub-cluster. • ALTER: allows users to modify the attributes of a specified object. • DROP: allows users to delete a specified object.

- `has_nodegroup_privilege(nodegroup, privilege)`
Description: Checks whether a user has the 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 7-45 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	User	Existing username or ID
sequence	text, oid	Sequence	Existing sequence name or ID

Parameter	Valid Input Parameter Type	Description	Value Range
privilege	text	Permission	• USAGE: For sequences, allows users to use the nextval function. • SELECT: allows users to create a sequence. • UPDATE: allows users to execute the UPDATE statement. • ALTER: allows users to modify the attributes of a specified object. • DROP: allows users to delete a specified object. • COMMENT: allows users to define or modify comments of a specified object.

- **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**, **REFERENCESTRIGGER**, **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:

```
openGauss=# 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
openGauss=# SELECT has_table_privilege('tt', 'select');
has_table_privilege
-----
t
(1 row)

# View the database username list.
openGauss=# \du;
Role name |                               List of roles                               |
Member of | Attributes |
-----+-----+-----
omm       | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Operatoradmin, Policyadmin, UseFT | {}
simple_user |                               | {}
test      | Sysadmin | {}

openGauss=# SELECT has_table_privilege('omm', 'tt', 'select,INSERT WITH GRANT OPTION ');
has_table_privilege
-----
t
(1 row)
```

- **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 7-46 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name	User	Existing username

Parameter	Valid Input Parameter Type	Description	Value Range
privilege	text	ANY permission	Available values: CREATE ANY TABLE [WITH ADMIN OPTION] ALTER ANY TABLE [WITH ADMIN OPTION] DROP ANY TABLE [WITH ADMIN OPTION] SELECT ANY TABLE [WITH ADMIN OPTION] INSERT ANY TABLE [WITH ADMIN OPTION] UPDATE ANY TABLE [WITH ADMIN OPTION] DELETE ANY TABLE [WITH ADMIN OPTION] CREATE ANY SEQUENCE [WITH ADMIN OPTION] CREATE ANY INDEX [WITH ADMIN OPTION] CREATE ANY FUNCTION [WITH ADMIN OPTION] EXECUTE ANY FUNCTION [WITH ADMIN OPTION] CREATE ANY PACKAGE [WITH ADMIN OPTION] EXECUTE ANY PACKAGE [WITH ADMIN OPTION] CREATE ANY TYPE [WITH ADMIN OPTION]

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. For example, to list the names of all visible tables, run the following command:

```
openGauss=# 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.

```
openGauss=# 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 a given OID exists.

Return type: Boolean

```
openGauss=# 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 the definition of a constraint.

Return type: text

- `pg_get_constraintdef(constraint_oid, pretty_bool)`

Description: Obtains the 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. It can be useful when the content of system catalogs is checked. If the expression might contain Vars, specify the OID of the relationship they refer to as the second parameter; if no Vars are expected, zero is sufficient.
- `pg_get_functiondef(func_oid)`
Description: Obtains the definition of a function.
Return type: text
Example:

```
openGauss=# 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 the parameter list of the 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 required to identify a function, in the form it would need to appear in **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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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)
openGauss=# 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

```
openGauss=# 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:

```
openGauss=# 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)
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:

```
openGauss=# 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)              +  
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:

```
openGauss=# SELECT pg_typeof(33);
pg_typeof
-----
integer
(1 row)

openGauss=# SELECT typen FROM pg_type WHERE oid = pg_typeof(33);
typen
-----
4
(1 row)
```

- **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:

```
openGauss=# 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 parameter is not of a collatable data type, then an error is thrown.

- **getdistributekey(table_name)**

Description: Obtains a distribution key for a hash table.

Return type: text

Example:

```
openGauss=# SELECT getdistributekey('item');
getdistributekey
-----
i_item_sk
(1 row)
```

- **pg_get_serial_sequence(tablename, colname)**

Description: Obtains the sequence of the corresponding table name and column name.

Return type: text

Example:

```
openGauss=# 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:

```
openGauss=# select * from pg_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:

```
openGauss=# 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:

```
openGauss=# 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 7-47](#) describes its components.

Table 7-47 Snapshot components

Name	Description
xmin	Earliest XID (txid) that is still active. All earlier transactions will either be committed and visible, or rolled back.
xmax	First as-yet-unassigned txid . 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 txids at the time of the snapshot. The list includes only those active txids between xmin and xmax ; there might be active txids higher than xmax . A txid that is greater than or equal to xmin and less than xmax 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 txids 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 (**gxid**) 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, timestampz, 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, timestampz, 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 return values are as follows:

Table 7-48 `pgxc_stat_get_wal_senders` parameter description

Column Name	Description
<code>nodename</code>	Instance name
<code>sender_pid</code>	PID of the thread for sending logs

Column Name	Description
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
sender_replay_location	Log location of the current instance. If the DN is a primary DN, the location is the value of sender_flush_location . 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 7-49 pgxc_stat_get_wal_senders_status parameter description

Column Name	Description
nodename	Name of the primary node
source_ip	IP address of the primary node
source_port	Port number of the primary node
dest_ip	IP address of the standby node
dest_port	Port number of the standby node

Column Name	Description
sender_pid	PID of the sender 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
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 7-50 `gs_get_next_xid_csn` parameter description

Column Name	Description
nodename	Node name.
next_xid	ID of the next transaction on the current node.
next_csn	Next CSN of the current node.

- `simsearch_lib_load_status()`
Description: Queries the dynamic library loading status (success or failure).
Return type: SETOF record
- `simsearch_gpu_vector_status()`
Description: Queries whether there is a vector in the status of searchlet.
Return type: SETOF record

** NOTE**

This function is no longer supported in the current version due to specification changes. Do not use this function.

- `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_delta_info`
Description: Obtains delta information from a single DN.
Parameter: rel text, schema_name text
Return type: part_name text, live_tuple bigint, data_size bigint, blocknum bigint
 - `pgxc_get_delta_info`
Description: Obtains delta information from all DNs. Only users with the SYSADMIN or MONADMIN permission can access the information.
Parameter: rel text, schema_name text
Return type: part_name text, live_tuple bigint, data_size bigint, blocknum bigint
 - `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
 - `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, 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_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, 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, 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, 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: pronom name, pronomspace oid, proowner oid, proleng 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, prorettype 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 7-51 `get_gtm_lite_status` parameter description

Column Name	Description
xmin	Minimum active XID in the GTM.

Column Name	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 7-52 `get_gtm_lite_status` parameter description

Column Name	Description
backup_xid	Backup XID in the GTM.
csn	Latest CSN issued by the GTM.

7.5.23 System Administration Functions

7.5.23.1 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:

```
openGauss=# 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**, the 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 SET statement.

Example:

```
openGauss=# SELECT set_config('log_statement_stats', 'off', false);
```

```
set_config
```

```
-----  
off
```

```
(1 row)
```

7.5.23.2 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. Use a relative path for files in the database cluster directory, and a path matching the **log_directory** configuration setting for log files. Only the database initial user 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:

```
openGauss=# 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:

```
openGauss=# 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: This function is similar to the **pg_read_file** function, but the return value is of the bytea type. The encoding check will not be performed. This function can be used together with the **convert_from** function to read a file in a specified encoding.

```
openGauss=# 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 use cases are as follows:

```
openGauss=# SELECT * FROM pg_stat_file('filename');
openGauss=# SELECT (pg_stat_file('filename')).modification;
```

Example:

```
openGauss=# 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)
openGauss=# 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)
openGauss=# SELECT (pg_stat_file('postmaster.pid')).modification;
modification
-----
2017-06-01 17:18:08+08
(1 row)
```

7.5.23.3 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 exists.

- `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.

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. This function can be used only by the initial user.

- `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.

NOTICE

This function can terminate non-thread pool threads and active thread pool threads, but cannot terminate inactive thread pool threads.

Example:

```
openGauss=# SELECT pid from pg_stat_activity;
           pid
-----
140657876268816
(1 rows)

openGauss=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend
-----
t
(1 row)
```

7.5.23.4 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. (**operation_mode** can be enabled only by SYSADMIN, replication role, or OPRADMIN.) 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.

```
openGauss=# SELECT pg_start_backup('label_goes_here',true);
pg_start_backup
-----
0/3000020
(1 row)
```

- **pg_stop_backup()**

Description: Completes online backup. **operation_mode** can be enabled only by SYSADMIN, replication role, or OPRADMIN.

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. In this way, 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. You need to enable **operation_mode** as an administrator or O&M administrator.

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:

```
openGauss=# 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_tracked_location()`

Description: Queries the LSN location 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 records 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 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. You need to enable **operation_mode** as an administrator or O&M administrator.

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. You need to enable **operation_mode** as an administrator or O&M administrator.

Return type: records

- `pg_enable_delay_xlog_recycle()`

Description: Enables the Xlog recycling delay function for CN recovery. You need to enable **operation_mode** as an administrator or O&M administrator.

Return type: void

- `pg_disable_delay_xlog_recycle()`

Description: Disables the Xlog recycling delay function for CN recovery. You need to enable **operation_mode** as an administrator or O&M administrator.

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 location 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 enable 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 and returns the log range where DDL delay takes effect. 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 disable DDL statements with delay.

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 **true**.

Return type: text. The content is the location of the segment log.

- `gs_block_dw_io(timeout int, identifier text)`

Description: Blocks dual-write page flushing.

Parameter description:

- `timeout`

Block duration.

Value range: [0,3600] (s). 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 dual-write 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**.

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, 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.

7.5.23.5 Dual-Cluster DR Control Functions

Dual-cluster DR control functions can be used to create an archive slot, which specifies the OBS information for storing physical logs.

- `pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)`

Description: Creates an OBS or a NAS archive slot. **slotname** indicates the slot name of the DR standby node. The primary and standby nodes must use the same slot name. If **dummy_standby** is set to **false**, one primary and multiple standby nodes are deployed. **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, where **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 **i**, and **is_recovery** indicates whether the slot is an archive slot or a recovery slot. The value **0** indicates that the slot is an archive slot and is used by the primary database instance. The value **1** indicates that the slot is a recovery slot and is used by the database instance for DR. **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 specified, 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** of the current standby for DR.

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.

Example:

Create an OBS archive slot.

```
openGauss=# select * from pg_create_physical_replication_slot_exten('uuid', false, 'OBS;obs.cn-
north-7.ulanqab.huawei.com;dyk;19D772JBCACXX3KWS51D;*****;openGauss_uuid/dn1;0;0', false);
 slotname | xlog_position
-----+-----
 uuid    |
(1 row)
```

Create a NAS archive slot.

```
openGauss=# select * from pg_create_physical_replication_slot_exten('uuid', false, 'NAS;/data/nas/
media/openGauss_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 in the cluster for DR and can be deleted on OBS.

Return type: `xlog_file_name` text, indicating the name of the log file that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

```
openGauss=# select gs_set_obs_delete_location('0/54000000');
 gs_set_obs_delete_location
-----
 0000000100000000000000054_00
(1 row)
```

- `gs_hadr_do_switchover()`

Description: Truncates services during a planned switchover in the primary cluster in the remote DR scenario.

Return type: Boolean, indicating whether the service truncation is successful and whether the switchover process can be performed normally.

- `gs_set_obs_delete_location_with_slotname(cstring, cstring )`

Description: Sets the location where OBS archived logs can be deleted in a DR relationship. The first parameter indicates the LSN. The logs before this location have been replayed and flushed to disks in the standby database instance for DR and can be deleted on OBS. The second parameter indicates the name of the archive slot.

Return type: `xlog_file_name` text, indicating the name of the log file that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

- `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.

7.5.23.6 Dual-Cluster DR Query Functions

- **gs_get_global_barrier_status()**
Description: If two-city 3DC DR is enabled, the primary cluster and standby cluster for DR synchronize logs through OBS. The barrier log is flushed to disks in the primary cluster, and replayed in the standby cluster for DR to determine the archived log progress of the primary cluster and the log replay progress of the standby cluster for DR. **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: indicates the globally latest barrier ID.
global_achive_barrier_id: indicates the globally latest archived barrier ID.
- **gs_get_local_barrier_status()**
Description: If two-city 3DC DR is enabled, the primary cluster and standby cluster for DR synchronize logs through OBS. The barrier log is flushed to disks in the primary cluster, and replayed in the standby cluster for DR to determine the archived 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.
- **gs_get_global_barriers_status()**
Description: If two-city 3DC DR solutions based on OBS are enabled, logs of the primary database instance and multiple standby database instances for DR are synchronized through OBS. The barrier log is flushed to the disk of the primary database instance. The progress of archiving logs of the primary database instance and the progress of replaying logs of the standby database instances for DR are determined by replaying the standby database instances for DR. **gs_get_global_barriers_status** is used to query the latest global barrier that has been archived in OBS for the primary database instance.
Return type: text
slot_name: name of the slot used for DR.
global_barrier_id: globally latest barrier ID.
global_achive_barrier_id: globally latest archived barrier ID.
- **gs_upload_obs_file('slot_name', 'src_file', 'dest_file')**
Description: Function used by the primary cluster to upload data to OBS if two-city 3DC DR is enabled.
Return type: void
slot_name: name of the replication slot created by the CN in the primary cluster.

src_file: location of files to be uploaded in the CN data directory of the primary cluster.

dest_file: location of files uploaded to OBS.

- `gs_download_obs_file('slot_name', 'src_file', 'dest_file')`

Description: Function used by the standby cluster for DR to download data from OBS to the localhost if two-city 3DC DR is enabled.

Return type: void

slot_name: name of the replication slot created by the CN in the standby cluster for DR.

src_file: location of files to be downloaded from OBS.

dest_file: location of downloaded files in the CN data directory of the standby cluster for DR.

- `gs_get_obs_file_context('file_name', 'slot_name')`

Description: Queries file content on OBS if two-city 3DC DR is enabled.

Return type: text

file_name: name of the file on OBS.

slot_name: name of the replication slot created by the CN in the primary or standby cluster for DR.

- `gs_set_obs_file_context('file_name', 'file_context', 'slot_name')`

Description: Creates a file on OBS and writes content into the file if two-city 3DC DR is enabled.

Return type: text

file_name: name of the file on OBS.

file_context: content to be written into the file.

slot_name: name of the replication slot created by the CN in the primary cluster or standby cluster for DR.

- `gs_get_hadr_key_cn()`

Description: Creates a file on OBS and writes content into the file if two-city 3DC DR is enabled.

Return type: text

file_name: name of the file on OBS.

file_context: content to be written into the file.

slot_name: name of the replication slot created by the CN in the primary cluster or standby cluster for DR.

- `gs_hadr_has_barrier_creator()`

Description: Checks whether the **barrier_creator** thread exists on the current CN if two-city 3DC DR is enabled. If the thread exists, **true** is returned (the SYSADMIN permission is required).

Return type: Boolean

Note: This function is used only when a planned switchover is performed in the standby cluster for DR.

- `gs_hadr_in_recovery()`

Description: Checks whether the current node is in barrier-based log restoration if two-city 3DC DR is enabled. If the current node is in barrier-

based log 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

 NOTE

This function is used only when a planned switchover is performed in the standby cluster for DR.

- `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 (the SYSADMIN permission is required).

Return type: Boolean

Note: This function is used only when a planned switchover is performed on the standby database instance for DR in streaming DR solutions.

- `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 standby database instance for DR in the switchover process can be started only after the logs are sent (the SYSADMIN permission is required).

Return type: Boolean

Note: This function is used only when a planned switchover is performed on the standby database instance for DR.

- `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.)

Return type: record. The types and meanings of the fields are as follows:

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.

Parameter	Type	Description
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, the 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, the RPO time between the target primary and standby database instances (unit: second).
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μ 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: μ s) required for inserting Xlogs on the host to reach the specified RPO.

- 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.)

Return type: record. The types and meanings of the fields are as follows:

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.

Parameter	Type	Description
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, the 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, the RPO time between the target primary and standby database instances (unit: second).
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μ 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: μ s) required by xlogInsert on the host to reach the specified RPO.

7.5.23.7 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**.

- **pg_export_snapshot_and_csn()**

Description: Saves the current snapshot and returns its identifier. Compared with **pg_export_snapshot()**, **pg_export_snapshot()** returns a CSN, indicating the CSN of the current snapshot.

Return type: text

7.5.23.8 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.

```
openGauss=# 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:

```
openGauss=# SELECT pg_database_size('testdb');
pg_database_size
-----
51590112
(1 row)
```

- **pg_relation_size(oid)**

Description: Abbreviation of **pg_relation_size (... , 'main')**, specifying the disk space used by the table with a specified OID or index.

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 or an index, 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 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 data. The table name can be schema-qualified.
Return type: `bigint`
Note: `pg_total_relation_size` receives the OID or name of a table, and returns the sizes of the data and related indexes 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** is used to calculate the space of an independent data value.

Example:

```
openGauss=# SELECT datalength(1);
datalength
-----
4
(1 row)
```

The following table lists the supported data types and calculation methods.

Data Type			Storage Space
Numeri c data types	Integ er types	TINYINT	1
		SMALLINT	2
		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.
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

Description: pg_relation_filenode receives the OID or name of a table, index, or sequence, and returns the **filenode** number 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

7.5.23.9 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 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** is similar to **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 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

7.5.23.10 Logical Replication Functions

- `pg_create_logical_replication_slot('slot_name', 'plugin_name')`
Description: Creates a logical replication slot.
Parameter description:
 - `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 only **mppdb_decoding**

Return type: name, text

Note: The first return value is the slot name, and the second one is the start LSN for decoding in the logical replication slot. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the permissions of the built-in role gs_role_replication.

- pg_create_physical_replication_slot('slot_name', 'isDummyStandby')

Description: Creates a physical replication slot.

Parameter description:

- 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

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.

- pg_drop_replication_slot('slot_name')

Description: Deletes a streaming replication slot.

Parameter description:

- 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.

- pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

Description: Performs decoding but does not go to the next streaming replication slot. (The decoded result will be returned again during the next decoding.)

Parameter description:

- 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.

- LSN

Indicates a target LSN. Decoding is performed only when an LSN is less than or equal to this value.

Value range: a string, in the format of *xlogid/xrecoff*, for example, '1/2AAFC60'. (If this parameter is set to **NULL**, the target LSN indicating 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.

- **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

- include-xids

Specifies whether the decoded **data** column contains XID information.

Valid value: **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.

Valid value: **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.

Valid value: **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.
Valid value: **0** and **1**. The default value is **0**.
 - **0**: The decoding 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. The following is an example:

```
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 disks.
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 disks.
Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

Return type: text, xid, text

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, respectively. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the permissions of the built-in role `gs_role_replication`.

- `pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding and goes to the next 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`.

- `pg_logical_slot_peek_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')`

Description: Performs decoding in binary format and does not go to the next streaming replication slot. (The decoded data can be obtained again during the next decoding.)

Parameter description:

- `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.

- `LSN`

Indicates a target LSN. Decoding is performed only when an LSN is less than or equal to this value.

Value range: a string, in the format of *xlogid/xrecoff*, for example, `'1/2AAFC60'`. (If this parameter is set to **NULL**, the target LSN indicating 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.

- **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

- `include-xids`

Specifies whether the decoded **data** column contains XID information.

Valid value: **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.

Valid value: **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.
Valid value: **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. The following is an example:

```
select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list',
'public.t1,public.t2');
```

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 the permissions of the built-in role gs_role_replication.

- **pg_logical_slot_get_binary_changes**('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

Description: Performs decoding in binary format and does not go to the next 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.

- `pg_replication_slot_advance` ('slot_name', 'LSN')

Description: Directly goes to the streaming replication slot for a specified LSN, without outputting any decoded result.

Parameter description:

- `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.

- `LSN`

Indicates a target LSN. Next decoding will be performed only in transactions whose commit position is greater than this value. 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, in the format of *xlogid/xrecoff*

Return type: name, text

Note: A return result contains the slot name and LSN that is actually used for decoding. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the permissions of the built-in role `gs_role_replication`.

- `pg_get_replication_slots()`

Description: Obtains the replication slot list.

Example:

```
openGauss=# select * from pg_get_replication_slots();
 slot_name | plugin      | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn |
 dummy_standby | confirmed_flush
-----+-----+-----+-----+-----+-----+-----+-----+
dn_s1      |             | physical | 0      | t      |      |              | 0/23DB14E0 | f      |
slot1      | mppdb_decoding | logical  | 16304  | f      |      |              | 0/23DA5700 | f      |
(2 rows)
```

Return value: text, text, text, oid, Boolean, xid, xid, text, Boolean, text

- `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 when no DDL operation is performed.

The constraints are as follows:

1. When the API is called, only when **wal_level** is set to **logical**, the generated log files can be parsed. If the used Xlog file is not of the logical level, the decoded content does not have the corresponding value and type, and there is no other impact.
2. The Xlog file can be parsed only by a copy of a fully homogeneous DN to ensure that the metadata corresponding to the data can be found and no DDL or VACUUM FULL operation is performed.
3. You can find the Xlog to be parsed.

4. Do not read too many Xlog files at a time. You are advised to read one Xlog file at a time. It is estimated that the memory occupied by one Xlog file is two to three times the size of the Xlog file.

5. The Xlog file before scale-out cannot be decoded.

Note: Users who call this function must have the **SYSADMIN** permission or the O&M administrator permission in O&M mode.

Parameter description:

– LSN_start

Specifies the LSN at the start of decoding.

Value range: a string, in the format of *xlogid/xrecoff*, for example, '1/2AAFC60'. If this parameter is set to NULL, the start position of decoding logs is not limited.

– LSN_end

Specifies the LSN at the end of decoding.

Value range: a string, in the format of *xlogid/xrecoff*, for example, '1/2AAFC60'. (If this parameter is set to **NULL**, the target LSN indicating 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:

```
openGauss=# SELECT pg_current_xlog_location();
pg_current_xlog_location
-----
0/E62E238
(1 row)

openGauss=# create table t1 (a int primary key,b int,c int);
```

```
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for table "t1"
CREATE TABLE
openGauss=# insert into t1 values(1,1,1);
INSERT 0 1
openGauss=# insert into t1 values(2,2,2);
INSERT 0 1

openGauss=# select data from pg_logical_get_area_changes('0/
E62E238',NULL,NULL,'sql_decoding',NULL);
 location | xid | data
-----+-----+-----
0/E62E8D0 | 27213 | COMMIT (at 2022-01-26 15:08:03.349057+08) 3020226
0/E6325F0 | 27214 | COMMIT (at 2022-01-26 15:08:07.309869+08) 3020234
.....
```

- `gs_get_parallel_decode_status()`

Description: Monitors the length of the read log queue and decoding result queue of each decoder thread to locate the concurrent decoding performance bottleneck.

Return type: text, int, text, text, text, int64, int64

Example:

```
openGauss=# 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
-----+-----+-----+-----+-----+-----+-----
 slot1      |          2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748 | 0/1DCE2578
 |          42 | 192927504
(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, and **working_txn_memory** indicates the total memory (in bytes) occupied by the concatenation transactions in the sender thread.

- `pg_replication_origin_create (node_name)`

Description: Creates a replication source with a given external name and returns the internal ID assigned to it.

Note: The user who calls this function must have the SYSADMIN permission.

Parameter description:

- `node_name`

Specifies the name of the replication source to be created.

Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

Return type: oid

- `pg_replication_origin_drop (node_name)`

Description: Deletes a previously created replication source, including any associated replay progress.

Note: The user who calls this function must have the SYSADMIN permission.

Parameter description:

- node_name
Specifies the name of the replication source to be deleted.
Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).
- pg_replication_origin_oid (node_name)
Description: Searches for a replication source by name and returns the internal ID. If no such replication source is found, an error is thrown.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - node_name
Specifies the name of the replication source to be queried.
Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).Return type: oid
- pg_replication_origin_session_setup (node_name)
Description: Marks the current session for replaying from a given origin, allowing you to track replay progress. This parameter can be used only when no origin is selected. Run the **pg_replication_origin_session_reset** command to cancel the configuration.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - node_name
Specifies the name of the replication source.
Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).
- pg_replication_origin_session_reset ()
Description: Cancels the **pg_replication_origin_session_setup()** effect.
Note: The user who calls this function must have the SYSADMIN permission.
- pg_replication_origin_session_is_setup ()
Description: Returns a true value if a replication source is selected in the current session.
Note: The user who calls this function must have the SYSADMIN permission.
Return type: Boolean
- pg_replication_origin_session_progress (flush)
Description: Returns the replay position of the replication source selected in the current session.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - flush
Determines whether the corresponding local transaction has been flushed to disk.
Value range: BooleanReturn type: LSN

- **pg_replication_origin_xact_setup** (origin_lsn, origin_timestamp)
Description: Marks the current transaction as recommitted at a given LSN and timestamp. This function can be called only when **pg_replication_origin_session_setup** is used to select a replication source.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - **origin_lsn**
Position for replaying the replication source.
Value range: LSN
 - **origin_timestamp**
Time point when a transaction is committed
Value range: timestamp with time zone
- **pg_replication_origin_xact_reset** ()
Description: Cancels the **pg_replication_origin_xact_setup()** effect.
Note: The user who calls this function must have the SYSADMIN permission.
- **pg_replication_origin_advance** (node_name, lsn)
Description:
Sets the replication progress of a given node to a given position. This is primarily used to set the initial position, or to set a new position after a configuration change or similar change.
Note: Improper use of this function may cause inconsistent replication data.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - **node_name**
Specifies the name of an existing replication source.
Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).
 - **lsn**
Position for replaying the replication source.
Value range: LSN
- **pg_replication_origin_progress** (node_name, flush)
Description: Returns the position for replaying the given replication source.
Note: The user who calls this function must have the SYSADMIN permission.
Parameter description:
 - **node_name**
Specifies the name of the replication source.
Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).
 - **flush**
Determines whether the corresponding local transaction has been flushed to disk.
Value range: Boolean

- `pg_show_replication_origin_status()`
Description: Displays the replication status of the replication source.
Note: The user who calls this function must have the SYSADMIN permission.
Return type:
 - **local_id**: OID, which specifies the ID of the replication source.
 - **external_id**: text, which specifies the name of the replication source.
 - **remote_lsn**: LSN of the replication source.
 - **local_lsn**: local LSN.

7.5.23.11 Segment-Page Storage Functions

- `local_segment_space_info(tablename TEXT, databasename TEXT)`
Description: Generates usage information about all extent groups in the tablespace of the current node.

Return type:

node_name	Node name
extent_size	Extent specifications of an extent group. The unit is the number of blocks.
forknum	Fork number
total_blocks	Total number of extents in a physical file
meta_data_blocks	Number of blocks occupied by the metadata managed in a tablespace, including the space header and map page but excluding the segment head
used_data_blocks	Number of extents used for storing data, including the segment head
utilization	Percentage of the number of used blocks to the total number of blocks, that is, (the value of used_data_blocks + the value of meta_data_block)/the value of total_blocks
high_water_mark	High-water mark, indicating the number of allocated extents and maximum physical page number. Blocks that exceed the high-water mark are not used and can be directly recycled.

Example:

```
select * from local_segment_space_info('pg_default', 'testdb');
 node_name | extent_size | forknum | total_blocks | meta_data_blocks | used_data_blocks | utilization | high_water_mark
-----+-----+-----+-----+-----+-----+-----+-----
dn_6001_6002_6003 | 1 | 0 | 16384 | 4157 | 1 | .253784 | 4158
dn_6001_6002_6003 | 8 | 0 | 16384 | 4157 | 8 | .254211 |
```

4165
(2 rows)

- `global_segment_space_info`(tablespacename TEXT, databasename TEXT)

Description: Returns the usage information of all nodes in the cluster. The effect is similar to that of **local_segment_space_info**.

- `pg_stat_segment_extent_usage`(int4 tablespace oid, int4 database oid, int4 extent_type, int4 forknum)

Description: Specifies the usage information of each allocated extent in an extent group returned each time. **extent_type** indicates the type of the extent group. The value is an integer ranging from 1 to 5. If the value is not within the range, an error is reported. **forknum** indicates the fork number. The value is an integer ranging from 0 to 4. Currently, only the following values are valid: **0** for data files, **1** for FSM files, and **2** for visibility map files.

Return type:

Name	Description
start_block	Start physical page number of an extent
extent_size	Size of an extent
usage_type	Usage type of an extent, for example, segment head and data extent
owner_location	Object location of an extent to which a pointer points. For example, the owner of a data extent is the head of the segment to which the data extent belongs.
special_data	Position of an extent in its owner. The value of this field is related to the usage type. For example, special data of a data extent is the extent ID in the segment to which the data extent belongs.

The value of **usage_type** is enumerated. The meaning of each value is as follows:

- **Non-bucket table segment head**: data segment head of a non-hash bucket table
- **Non-bucket table fork head**: Fork segment header of a non-segment-page table
- **Bucket table main head**: main table segment header of a hash bucket table
- **Bucket table map block**: Map block of a hash bucket table
- **Bucket segment head**: segment head of each bucket in a hash bucket table
- **Data extent**: data block

Example:

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace where
spcname='pg_default'), (select oid::int4 from pg_database where datname='testdb'), 1, 0);
```

start_block	extent_size	usage_type	lower_location	special_data
4157	1	Bucket table main head	4294967295	0
4158	1	Bucket table map block	4157	0
4159	1	Bucket table map block	4157	1
4160	1	Bucket table map block	4157	2
4161	1	Bucket table map block	4157	3
4162	1	Bucket table map block	4157	4
4163	1	Bucket table map block	4157	5
4164	1	Bucket table map block	4157	6
4165	1	Bucket table map block	4157	7
4166	1	Bucket table map block	4157	8

- **local_space_shrink**(tablespacename TEXT, databasename TEXT)
Description: Shrinks specified physical segment-page space on the current node. Only the currently connected database can be shrunk.
Return value: empty
- **gs_space_shrink**(int4 tablespace, int4 database, int4 extent_type, int4 forknum)
Description: Works similar to **local_space_shrink**. That is, shrinks specified physical segment-page space. However, the parameters are different. The input parameters are the OIDs of the tablespace and database, and the value of **extent_type** is an integer ranging from 2 to 5. Note: The value **1** of **extent_type** indicates segment-page metadata. Currently, the physical file that contains the metadata cannot be shrunk. This function is used only by tools. You are advised not to use it directly.
Return value: empty
- **global_space_shrink**(tablespacename TEXT, databasename TEXT)
Description: Compresses segment-page storage space on all DN's in a cluster. This is performed on a CN.
Note: If **global_space_shrink** locks the cluster. DDL operations cannot be performed. **local_space_shrink** does not lock the cluster.
- **pg_stat_remain_segment_info**()
Description: Displays residual extents on the current node due to faults. Residual extents are classified into two types: segments that are allocated but not used and extents that are allocated but not used. The main difference is that a segment contains multiple extents. During reclamation, all extents in the segment need to be recycled.

Return type:

Name	Description
space_id	Tablespace ID
db_id	Database ID
block_id	Extent ID
type	Extent type. The options are as follows: ALLOC_SEGMENT , DROP_SEGMENT , and SHRINK_EXTENT .

The values of **type** are described as follows:

- **ALLOC_SEGMENT**: When a user creates a segment-page table and the segment is just allocated but the transaction of creating a table is not committed, the node is faulty. As a result, the segment is not used after being allocated.
- **DROP_SEGMENT**: When a user deletes a segment-page table and the transaction is successfully committed, the bit corresponding to the segment page of the table is not reset and a fault, such as power failure, occurs. As a result, the segment is not used or released.
- **SHRINK_EXTENT**: When a user shrinks a segment-page table and does not release the idle extent, a fault, such as power failure, occurs. As a result, the extent remains and cannot be reused.

Example:

```
select * from pg_stat_remain_segment_info();
space_id | db_id | block_id | type
-----+-----+-----+-----
1663     | 16385 | 4156    | ALLOC_SEGMENT
```

- **pg_free_remain_segment(int4 spaceId, int4 dbId, int4 segmentId)**
Description: Releases a specified residual extent. The value must be obtained from the **pg_stat_remain_segment_info** function. The function verifies input values. If the specified extent is not among the recorded residual extents, an error message is returned. If the specified extent is a single extent, the extent is released independently. If it is a segment, the segment and all extents in the segment are released.
Return value: empty

7.5.23.12 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

- 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.

```
openGauss=# 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.

```
openGauss=# 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:

```
openGauss=# 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**. Especially, in scenarios 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:

```
openGauss=# 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 current 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 metadata 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

- `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 by 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 7-53](#).
- 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 **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 7-53 Error table `public.pgxc_copy_error_log`

Column	Type	Description
relname	character varying	Table name, in the form of <i>Schema name.Table name</i> .
begintime	timestamp with time zone	Time when a data format error was reported
filename	character varying	Name of the source data file where a data format error occurs

Column	Type	Description
lineno	bigint	Number of the row where a data format error occurs in a source data file
rawrecord	text	Raw record of a data format error in the source data file
detail	text	Error details

- `pg_stat_get_data_senders()`
Description: Provides detailed information about the data-copy sending 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 7-54 generate_wdr_report parameter description

Parameter	Description	Range
<code>begin_snap_id</code>	Snapshot ID that starts the diagnosis report period.	N/A
<code>end_snap_id</code>	Snapshot ID that ends the diagnosis report period. By default, the value of end_snap_id is greater than that of begin_snap_id .	N/A
<code>report_type</code>	Specifies the type of the generated report.	• summary • detail • all: Both summary and detail types are included.
<code>report_scope</code>	Specifies the scope for a report to be generated.	• cluster: database-level information • node: node-level information.
<code>node_name</code>	• When report_scope is set to node, set this parameter to the name of the corresponding node. • If report_scope is set to cluster, this parameter can be omitted or set to NULL.	• node: node name in GaussDB. • cluster: This value is omitted, left blank or set to NULL.

- `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 7-55 reset_unique_sql parameter description

Parameter	Type	Description
scope	text	Clearance scope type. The options are as follows: 'GLOBAL' : Clears all CNs/DNs. If the value is 'GLOBAL' , this function can be executed only on the CN. 'LOCAL' : Clears the current node.
clean_type	text	'BY_USERID' : Unique SQL statements are cleared based on user IDs. 'BY_CNID' : Unique SQL statements are cleared based on CN IDs. 'ALL' : All data is cleared.
clean_value	int8	Clearance value corresponding to the clearance type. If the second parameter is set to ALL , 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 system administrators have the permission to execute this function.


```
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 eviction information.

Return type: record

- `local_candidate_stat()`

Description: Displays the number of pages in the candidate buffer chain of this instance and buffer eviction 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 7-56 `local_double_write_stat` parameters

Parameter	Type	Description
<code>node_name</code>	text	Instance name
<code>curr_dwn</code>	int8	Sequence number of the doublewrite file
<code>curr_start_page</code>	int8	Start page for restoring the doublewrite file
<code>file_trunc_num</code>	int8	Number of times that the doublewrite file is reused
<code>file_reset_num</code>	int8	Number of reset times after the doublewrite file is full
<code>total_writes</code>	int8	Total number of I/Os of the doublewrite file
<code>low_threshold_writes</code>	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.)
<code>high_threshold_writes</code>	int8	Number of I/Os for writing the doublewrite file with high efficiency (the number of I/O flushing pages at a time is more than 421.)
<code>total_pages</code>	int8	Total number of pages that are flushed to the doublewrite file area
<code>low_threshold_pages</code>	int8	Number of pages that are flushed with low efficiency
<code>high_threshold_pages</code>	int8	Number of pages that are flushed with high efficiency
<code>file_id</code>	int8	ID of the current doublewrite file

- `local_single_flush_dw_stat()`
Description: Displays the eviction 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_cgroup_map_ng_conf(group name)`
Description: Reads the Cgroup configuration file of a specified node group. Only users with the SYSADMIN permission can execute this function.
Return type: record
- `pgxc_cgroup_map_ng_conf(group name)`
Description: Reads the Cgroup configuration file of a specified node group on all nodes. Only users with the SYSADMIN permission can execute this function.
Return type: Boolean
- `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 location of the Xlog flushed from the current node.
Return type: text
- `pg_get_sync_flush_lsn()`
Description: Returns the location 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:

```
openGauss=# select locktag_decode('271b:0:0:0:0:6');
           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 7-57 `disable_conn` parameter description

Parameter	Type	Description
<code>disconn_mode</code>	text	DN connection mode: • 'prohibit_connection': rejects to connect to all DNs. • 'specify_connection': forcibly connects to a DN. • 'polling_connection': connects to all DNs in polling mode.
<code>host</code>	text	IP address of the DN
<code>port</code>	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 7-58 `db_perf.get_global_full_sql_by_timestamp` parameter description

Parameter	Type	Description
<code>start_timestamp</code>	timestamp with time zone	Start point of the SQL start time range.
<code>end_timestamp</code>	timestamp with time zone	End point of the SQL start time range.

- `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 7-59 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.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

- 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 7-60 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 plaintext .
pretty	boolean	Specifies whether to display the text in pretty format when format is set to plaintext . The options are as follows: - The value true indicates that \n is used to separate events. - The value false indicates that events are separated by commas (,).

- 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 collation 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_catalog_attribute_records()`
Description: Returns the definition of each column in a system catalog specified by OID. 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 7-61 Parameter description of `dynamic_func_control`

Parameter	Type	Description
scope	text	Scope where the function is to be dynamically enabled. Currently, only ' GLOBAL ' and ' LOCAL ' are supported.
function_name	text	Function name. Currently, only ' STMT ' is supported.

Parameter	Type	Description
action	text	When function_name is set to 'STMT', the value of action can only be TRACK, UNTRACK, LIST, or CLEAN. • TRACK: records the full SQL information of normalized SQL statements. • UNTRACK: cancels the recording of full SQL information of normalized SQL statements. • LIST: lists normalized SQL information that is recorded in the current track. • CLEAN: cleans normalized SQL information that is recorded in the current track.
params	text[]	When function_name is set to 'STMT', the parameters corresponding to different actions are set as follows: • TRACK: '{"Normalized SQLID", "L0/L1/L2"}' • UNTRACK: '{"Normalized SQLID"}' • LIST: '{}' • CLEAN: '{}'

- **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 SYSADMIN or OPRADMIN can execute this function.

Table 7-62 gs_parse_page_bypath parameters

Parameter	Type	Description
path	text	- For an ordinary table or a segment-page ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>. For example, base/16603/16394. - For a segment-page hash bucket table, the relative path is <i>Tablespace name/Database OID/Logical page number of the segment head_b(bucketid)</i>. For example, base/16603/16394_b1437. - You can run the pg_relation_filepath(table_name text) command to query the relative path of the table file. To obtain the path of the partitioned table, view the pg_partition system catalog and call pg_partition_filepath(partition_oid). - Valid path formats are as follows: - global/relNode - base/dbNode/relNode - pg_tblspc/spcNode/version_dir/dbNode/relNode For hash bucket tables, add _b(bucketid) to the end of the path.
blocknum	bigint	-1: Information about all blocks (forcibly parsed from disks) 0-MaxBlockNumber: Information about the corresponding block
relation_type	text	heap: Astore table btree: B-tree index segment: Segment-page table
read_memory	boolean	false: The system parses the page from the disk file. true: 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.

- 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 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 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 SYSADMIN or OPRADMIN can execute this function.

Table 7-63 `gs_xlogdump_tablepath` parameters

Parameter	Type	Description
path	text	• For an ordinary table or a segment-page ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>. For example, base/16603/16394. • For a segment-page hash bucket table, the relative path is <i>Tablespace name/Database OID/Logical page number of the segment head_b(bucketid)</i>. For example, base/16603/16394_b1437. • You can run the pg_relation_filepath(table_name text) command to query the relative path of the table file. To obtain the path of the partitioned table, view the pg_partition system catalog and call pg_partition_filepath(partition_oid). • Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode For hash bucket tables, add _b(bucketid) to the end of the path.
blocknum	bigint	• -1: Information about all blocks (forcibly parsed from disks) • 0-MaxBlockNumber: Information about the corresponding block

Parameter	Type	Description
relation_type	text	• heap: Astore table • btree: B-tree index • segment: Segment-page table

- **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 SYSADMIN or OPRADMIN can execute this function.

Table 7-64 gs_xlogdump_parsepage_tablepath parameters

Parameter	Type	Description
path	text	• For an ordinary table or a segment-page ordinary table, the relative path is <i>Tablespace name/Database OID/Relfilenode of the table (physical file name)</i>. For example, base/16603/16394. • For a segment-page hash bucket table, the relative path is <i>Tablespace name/Database OID/Logical page number of the segment head_b(bucketid)</i>. For example, base/16603/16394_b1437. • You can run the pg_relation_filepath(table_name text) command to query the relative path of the table file. To obtain the path of the partitioned table, view the pg_partition system catalog and call pg_partition_filepath(partition_oid). • Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode For hash bucket tables, add _b(bucketid) to the end of the path.

Parameter	Type	Description
blocknum	bigint	• -1: Information about all blocks (forcibly parsed from disks) • 0-MaxBlockNumber: Information about the corresponding block
relation_type	text	• heap: Astore table • btree: B-tree index • segment: Segment-page table
read_memory	boolean	• false: The system parses the page from the disk file. • true: 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.

- `gs_index_recycle_queue(Oid oid, int type, uint32 blkno)`
Description: Parses the UB-tree index recycling queue information.
Return type: record

Table 7-65 `gs_index_recycle_queue` parameters

Parameter	Type	Description
oid	Oid	• Index file relfilenode, which can be queried using select relfilenode from pg_class where relname='name', where <i>name</i> indicates the name of the index file.
type	int	• 0 indicates that the entire queue to be recycled is parsed. • 1 indicates that the entire empty page queue is parsed. • 2 indicates that a single page is parsed.
blkno	uint32	ID of the recycling queue page. This parameter is valid only when type is set to 2 . The value of blkno 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 WAL insertion status table in the Xlog.
Return type: record

Table 7-66 gs_stat_wal_entrytable parameters

Category	Parameter	Type	Description
Input parameter	idx	int64	• -1: queries all elements in an array. • 0-Maximum value: content of a specific array element.
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.
Output parameter	status	uint32	Specifies whether the Xlog corresponding to the current entry has been completely copied to the WAL buffer. • 0: Not copied. • 1: Copied

- gs_walwriter_flush_position()
Description: Outputs the refresh position of WALs.
Return type: record

Table 7-67 gs_walwriter_flush_position parameters

Category	Parameter	Type	Description
Output parameter	last_flush_status_entry	int32	Subscript 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.

Category	Parameter	Type	Description
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.
Output parameter	send_result	uint64	Xlog sending position on the current host.
Output parameter	shm_rqst_write_pos	uint64	Write position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.
Output parameter	shm_rqst_flush_pos	uint64	Flush position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.
Output parameter	shm_result_write_pos	uint64	Write position of the LogwrtResult request in the XLogCtl recorded in the shared memory.
Output parameter	shm_result_flush_pos	uint64	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 WALs, data volume, and Xlog file information.

Return type: record

Table 7-68 gs_walwriter_flush_stat parameters

Category	Parameter	Type	Description
Input parameter	operation	int	• -1: Disables the statistics function. (Default value) • 0: Enables the statistics function. • 1: Queries statistics. • 2: Resets statistics.
Output parameter	write_times	uint 64	Number of times that the Xlog calls the write API.
Output parameter	sync_times	uint 64	Number of times that the Xlog calls the sync API.
Output parameter	total_xlog_sync_bytes	uint 64	Total number of backend thread requests for writing data to Xlogs.
Output parameter	total_actual_xlog_sync_bytes	uint 64	Total number of Xlogs that call the sync API for disk flushing.
Output parameter	avg_write_bytes	uint 32	Number of Xlogs written each time the XLogWrite API is called.
Output parameter	avg_actual_write_bytes	uint 32	Number of Xlogs written each time the write API is called.
Output parameter	avg_sync_bytes	uint 32	Average number of Xlogs for requesting the sync API each time.
Output parameter	avg_actual_sync_bytes	uint 32	Actual number of Xlogs for disk flushing by calling the sync API each time.

Category	Parameter	Type	Description
Output parameter	total_write_time	uint64	Total time for calling the write API (unit: μ s).
Output parameter	total_sync_time	uint64	Total time for calling the sync API (unit: μ s).
Output parameter	avg_write_time	uint32	Average time for calling the write API each time (unit: μ s).
Output parameter	avg_sync_time	uint32	Average time for calling the sync API each time (unit: μ 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 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 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	timestampz	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 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

- `gs_undo_dump_xid(undo_xid xid)`
Description: Parses undo records based on the XID.
Return type: record

Table 7-69 `gs_undo_dump_xid` parameter description

Category	Parameter	Type	Description
Input parameter	undo_xid	xid	XID
Output parameter	undoptr	xid	Start position of the undo record to be parsed
Output parameter	xactid	text	XID
Output parameter	cid	text	Command ID
Output parameter	relid	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

Category	Parameter	Type	Description
Output parameter	oldxactid	text	Previous XID
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	Specifies whether to reserve the record.
Output parameter	flag	text	Flag 1
Output parameter	flag2	text	Flag 2
Output parameter	t_hoff	text	Length of the undo record data header

- `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

7.5.24 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: timestamptz
- `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 on this function in the current transaction, excluding the time spent on this function internally calling other functions.
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: `bigint`
- `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.
- `pg_stat_get_wal_senders()`
Description: Queries walsender information on the primary node.
Return type: `setofrecord`
The following table describes return columns.

Table 7-70 Return column description

Column	Type	Description
<code>pid</code>	<code>bigint</code>	Thread ID of the WAL sender
<code>sender_pid</code>	<code>integer</code>	Lightweight thread ID of the WAL sender
<code>local_role</code>	<code>text</code>	Type of the primary node
<code>peer_role</code>	<code>text</code>	Type of the standby node
<code>peer_state</code>	<code>text</code>	Status of the standby node
<code>state</code>	<code>text</code>	Status of the WAL sender
<code>catchup_start</code>	timestamp with time zone	Startup time of a catchup task
<code>catchup_end</code>	timestamp with time zone	End time of a catchup task
<code>sender_sent_location</code>	<code>text</code>	Sending position of the primary node

Column	Type	Description
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_group	text	Group to which the synchronous replication belongs
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 7-71 Return values

Return Parameter	Type	Description
local_role	text	Local role
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: Specifies the number of sequential row scans done if parameters are in a table or the number of index scans done 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 or MONADMIN permission can access the information.

Return type: text

Example:

```
openGauss=# 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_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 table partition.
Return type: bigint
 - `pg_stat_get_partition_dead_tuples(oid)`
Description: Specifies the number of dead rows in a table partition.
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_xact_partition_tuples_inserted(oid)`
Description: Specifies the number of inserted tuples in the active subtransactions 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 subtransactions 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 subtransactions 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 subtransactions related to a table partition.
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 7-72 Description

Return Parameter	Type	Description
nodename	name	Node name
nspname	name	Name of a namespace
relname	name	Name of an object, such as a table, an index, or a view
partname	name	Partition name
n_dead_tuples	bigint	Number of dead rows in a table partition
n_live_tuples	bigint	Number of live rows in a table partition
changes_since_analyze	bigint	Number of changes generated by ANALYZE

- `pg_autovac_status(oid)`

Description: Returns autovac information, such as **nodename**, **nspname**, **relname**, **analyze**, **vacuum**, thresholds of **analyze** and **vacuum**, 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 7-73 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 ANALYZE.
anltuples	bigint	Number of ANALYZE tuples
anlthresh	bigint	ANALYZE threshold
dovacuum	Boolean	Specifies whether to execute VACUUM.
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.
Return type: text
- `pg_stat_get_last_data_changed_time(oid)`
Description: Returns the time when INSERT, UPDATE, DELETE, or EXCHANGE/TRUNCATE/DROP PARTITION was performed last time on a table. The data in the **last_data_changed** column of the [PG_STAT_ALL_TABLES](#) view is calculated by using this function. The performance of obtaining the last modification time by using the view is poor when the table has a large amount of data. In this case, you are advised to use this function.
Return type: timestampz

- `pg_stat_set_last_data_changed_time(oid)`
Description: Manually changes the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** was last performed.
Return type: void
- `pg_backend_pid()`
Description: Specifies the thread ID of the server thread attached to the current session.
Return type: integer
- `pg_stat_get_activity(integer)`
Description: Returns a record of a background process with a specific PID. When the parameter is **NULL**, it returns records of all active background processes. The returned result does not contain the **connection_info** column. The initial user, system administrators and users with the MONADMIN permission can view all data. Common users can only query their own results.

Example:

```
openGauss=# select * from pg_stat_get_activity(139881386280704);
 datid |   pid   | sessionid | usesysid | application_name | state |
 query | waiting | xact_start | query_start |
 backend_start | state_change | client_addr | client_hostname | client_port | enqueue
 | query_id | srespool | global_sessionid | unique_sql_id | trace_id
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
16545 | 139881386280704 | 69 | 10 | gsql | active | select * from
pg_stat_get_activity(139881386280704); | f | 2022-01-18 19:43:05.167718+08 | 2022-01-18
19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 | | | -1 | |
72620543991624410 | default_pool | 1938253334#69#0 | 3751941862 |
(1 row)
```

Return type: setofrecord

The following table describes return parameters.

Table 7-74 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

Return Parameter	Type	Description
state	text	Overall status of the backend
query	text	Latest query at the backend. If state is active , 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 for a lock. If the backend is currently waiting for a lock, the value is true .
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 query_start column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if state is not active . 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.
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 state was last modified

Return Parameter	Type	Description
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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
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 background process with the specified PID. A record for each active background process 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 7-75 Return values

Return Value	Return 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 state is active , 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 for a lock. If the backend is currently waiting for a lock, the value is true .
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 query_start column.

Return Value	Return 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 state is not active . 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.
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 state was last modified
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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

Return Value	Return 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
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_ng(integer)`

Description: Returns a record about the active backend thread with the specified PID. A record for each active backend 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
node_group	text	Node group of the user to which the data belongs.

- `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 a specified server thread.
Return type: `bigint`
- `pg_stat_get_backend_dbid(integer)`
Description: Specifies the ID of a database connected to a specified server process.
Return type: `oid`
- `pg_stat_get_backend_userid(integer)`
Description: Specifies the user ID of a specified server process. This function can be called only by system administrators.
Return type: `oid`
- `pg_stat_get_backend_activity(integer)`
Description: Queries the current activity of a specified server process. The query result can be obtained only when the user who calls this function is a system administrator or the same user as that of the session being queried and **track_activities** is enabled.
Return type: `text`
- `pg_stat_get_backend_waiting(integer)`
Description: Returns a true value if a specified server process is waiting for a lock, but only when the user who calls this function is a system administrator or the same user as that of the session being queried and **track_activities** is enabled.
Return type: `Boolean`
- `pg_stat_get_backend_activity_start(integer)`
Description: Specifies the time when a specified server process's currently executing query is started only when the user who calls this function is a system administrator or the same user as that 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 a specified server process's currently executing transaction is started only when the user who calls this function is a system administrator or the same user as that 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 a specified server process is started. If the current user is neither a system administrator nor the same user as that 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 a backend connected to a specified client. 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 of a backend connected to a specified client. 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 (the SYSADMIN permission is required).
Return type: void
 - `gs_stat_reset()`
Description: Resets all statistics counters for the current database on each node to zero (the SYSADMIN permission is required).
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 (the SYSADMIN permission is required).
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 (the SYSADMIN permission is required).
Return type: void
 - `pg_stat_reset_single_function_counters(oid)`
Description: Resets statistics for a single function in the current database to zero (the SYSADMIN permission is required).
Return type: void
 - `pg_stat_get_db_cu_hdd_asyn(oid)`
Description: Obtains the times CU is asynchronously read from a disk by a database of the current node.
Return type: bigint
 - `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 terminated.
Return type: text
 - `fenced_udf_process(integer)`
Description: Shows the number of local UDF Master and Work processes. 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 node group to be queried.

For example, to query the Cgroup information for the 'installation' node group, 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 DN nodes in a node group. The unit is byte.

Return type: bigint

- gs_check_logic_cluster_consistency()

Description: Checks whether the system information of all node groups in the system is consistent. If no record is returned, the information is consistent. Otherwise, the node group information on CNs and DN nodes in a node group is inconsistent. This function cannot be called during redistribution in scale-in or scale-out.

Return type: record

- `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 or CUs 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 or CUs after all nodes in the cluster are started.

Return type: record

- `pg_stat_bad_block_clear()`

Description: Deletes the page and CU damage information that is read and recorded on the node (the SYSADMIN permission is required).

Return type: void

- `pgxc_stat_bad_block_clear`

Description: Deletes the page or CU damage information that is read and recorded on all nodes in the cluster (the SYSADMIN permission is required).

Return type: void

- `pgxc_log_comm_status(void)`

Description: When the TCP proxy communication is used, the PGXC system view exports the communication layer status of DN to each log file.

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 (Top , CLASS , BAKWD , DEFWD , or TSWD).
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: μ s).
resp_max	201891	Maximum response time of frontend transactions (unit: μ s).
resp_avg	43564	Average response time of frontend transactions (unit: μ s)

Attribute	Value	Description
resp_total	217822	Total response time of frontend transactions (unit: μ s).
bg_commit_counter	910	Number of backend transactions that were committed.
bg_rollback_counter	0	Number of backend transactions that were rolled back.
bg_resp_min	97	Minimum response time of backend transactions (unit: μ s).
bg_resp_max	678080687	Maximum response time of backend transactions (unit: μ s).
bg_resp_avg	327847884	Average response time of backend transactions (unit: μ s).
bg_resp_total	298341575300	Total response time of backend transactions (unit: μ 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: μ s)
CPU_TIME	311777	Total CPU time consumed by all threads (unit: μ s)
EXECUTION_TIME	380037	Total time consumed on the executor (unit: μ s)
PARSE_TIME	6033	Total time consumed for parsing SQL statements (unit: μ s)
PLAN_TIME	173356	Total time consumed for generating an execution plan (unit: μ s)
REWRITE_TIME	2274	Total time consumed for rewriting queries (unit: μ s)
PL_EXECUTION_TIME	0	Total time consumed for executing PL/pgSQL statements (unit: μ s)
PL_COMPILATION_TIME	557	Total time consumed for compiling SQL statements (unit: μ s)

Stat_name Attribute	Value	Description
NET_SEND_TIME	1673	Total time consumed for sending data over the network (unit: μ s)
DATA_IO_TIME	426622	Total time consumed for reading and writing data (unit: μ s)

- **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.
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 distribution information about the response time of 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
- **gs_paxos_stat_replication()**
Description: Queries the standby node information on the primary node. Currently, the distributed mode is not supported.
- **get_paxos_replication_info()**
Description: Queries the primary-standby replication information. Currently, the distributed mode is not supported.
- **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 7-76 Return values

Name	Type	Description
ngname	text	Node group name.
memorytype	text	Memory type. The value must be one of the following: • ng_total_memory: total memory size of the node group. • ng_used_memory: memory used by the node group. • ng_estimate_memory: estimated memory used by the node group. • ng_foreignrp_memsize: total memory of the external resource pool of the node group. • ng_foreignrp_usedsize: memory used by the external resource pool of the node group. • ng_foreignrp_peaksize: peak memory used by the external resource pool of the node group. • ng_foreignrp_mempct: percentage of the external resource pool of the node group to the total memory of the node group. • ng_foreignrp_estmsize: estimated memory used by the external resource pool of the node group.
memorybytes	integer	Size of allocated 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 operations completed per second

Name	Type	Description
write_per_second	float	Number of write operations completed 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_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 of the same file.

- 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.
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.

- 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.

- 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.
- 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 NULL , 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

```
openGauss=# 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 0 .
count	bigint	Frequency of accessing the hotspot key

- gs_stat_clean_hotkeys()

 NOTE

- Hotspot key detection is designed for high-concurrency and heavy-traffic scenarios. In the scenario where the access is performed for several times, the clearing 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

```
openGauss=# 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

```
openGauss=# 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

```
openGauss=# 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 communication 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 communication 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_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
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 that the client is connected via a Unix socket on the server machine or that 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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

Name	Type	Description
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 query_start column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if state is not active
state_change	timestamp with time zone	Time when state 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 true .
enqueue	text	Queuing state of a statement. Possible values are: • waiting in queue: The statement is in the queue. • Empty: The statement is being executed.

Name	Type	Description
state	text	Overall status of the backend Possible values are: • active: The background is executing a query. • idle: The background is waiting for a new client command. • idle in transaction: The background is in a transaction, but there is no statement being executed in the transaction. • idle in transaction (aborted): The background is in a transaction, but there are statements failed in the transaction. • fastpath function call: The background is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in the background. NOTE Only system administrators can view the session status of their accounts. The state information of other accounts is empty. For example, after user judy is connected to the database, the state information of user joe and the initial user omm in pgxc_stat_activity is empty. <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 state is active , 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
- **pv_session_memory**
Description: Collects statistics on memory usage at the session level, including all the memory allocated to GaussDB 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 7-77 Return values

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

Name	Type	Description
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.

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_start	timestampz	Time when the current query starts 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.
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.

Return type: SETOF record

Name	Type	Description
nodename	text	Name of the CN connected to the user session

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_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.
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

There are 17 status information 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_inmemory**, **sort_idisk**, **cu_mem_hit**, **cu_hdd_sync_read**, and **cu_hdd_asyread**.

- 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 GaussDB and stream threads on DN 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_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_stat_db_cu()

Description: Collects statistics on the CU hit ratio of all databases on each cluster node in a view. 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: Aggregates I/O statistics on 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_bgwriter_stat()`
Description: Aggregates statistics on the background writer 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 or SYSADMIN 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 distribution information about the response time of 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.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 collection result is cleared. When the input parameter is set to "", the collection function is disabled. Only the initial user or a user with the MONADMIN permission can execute 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. Only the initial user or a user with the MONADMIN permission can execute this function.

Return type: record

- DBE_PERF.global_io_wait_info()

Description: Queries real-time I/O control statistics on all CNs and DN.

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

- pg_stat_get_file_stat()

Description: Collects statistics on 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: Aggregates I/O statistics on 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](#).

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 [GLOBAL_THREADPOOL_STATUS](#).

Return type: record

- `comm_check_connection_status`

Description: Returns the connection status between the CN and all active nodes (CNs and primary DN). 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 DN). This function can be queried only on CNs. Permission control is inherited from the **DBE_PERF** schema.

Parameter: nan

Return type: node_name text, remote_name text, remote_host text, remote_port integer, is_connected Boolean, and no_error_occur Boolean

- remote_candidate_stat()

Description: Displays the number of pages in the candidate buffer chain of the instance and buffer eviction information, including the normal buffer pool and segment buffer pool.

Return type: record

Table 7-78 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 7-79 remote_ckpt_stat parameter description

Parameter	Type	Description
node_name	text	Instance name

Parameter	Type	Description
ckpt_redo_point	text	Checkpoint in 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 CSNlog flushing pages from the startup time to the current time
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 7-80 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 file with low efficiency (the number of I/O flushing pages at a time is less than 16.)

Parameter	Type	Description
high_threshold_writes	int8	Number of I/Os for writing the doublewrite file 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 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 7-81 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 doublewrite file
total_writes	bigint	Total number of data write pages in 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

- 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 7-82 remote_pagewriter_stat parameter description

Parameter	Type	Description
node_name	text	Instance name

Parameter	Type	Description
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
queue_head_page_rec_lsn	text	recovery_lsn of the first dirty page in the dirty page queue in the current instance
queue_rec_lsn	text	recovery_lsn of the dirty page queue in the current instance
current_xlog_insert_lsn	text	The write position of Xlogs in the current instance
ckpt_redo_point	text	Checkpoint in the current instance

- remote_recovery_status()

Description: Displays log flow control information about the primary and standby nodes (excluding the current node and not usable on DNs).

Return type: record

Table 7-83 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)

Parameter	Type	Description
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: μ s).

- remote_rto_stat()

Description: Displays log flow control information about the primary and standby nodes (excluding the current node and not usable on DNs).

Return type: record

Table 7-84 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 sleep time required by the primary node to achieve the expected flow control time (unit: μ s)

- 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 7-85 remote_redo_stat parameter description

Parameter	Type	Description
node_name	text	Instance name
redo_start_ptr	int8	Start point for replaying the current instance logs
redo_start_time	int8	Start time (UTC) when the current instance logs are replayed

Parameter	Type	Description
redo_done_time	int8	End time (UTC) when the current 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 current instance reads and replays logs
read_xlog_io_total_dur	int8	Total I/O latency when the current instance reads and replays logs
read_data_io_counter	int8	Number of data page I/O reads during replay in the current instance
read_data_io_total_dur	int8	Total I/O latency of data page reads during replay in the current instance
write_data_io_counter	int8	Number of data page I/O writes during replay in the current instance
write_data_io_total_dur	int8	Total I/O latency of data page writes during replay in the current instance
process_pending_counter	int8	Number of synchronization times of log distribution threads during replay in the current instance
process_pending_total_dur	int8	Total synchronization latency of log distribution threads during replay in the current instance

Parameter	Type	Description
apply_counter	int8	Number of synchronization times of replayer threads during replay in the current instance
apply_total_dur	int8	Total synchronization latency of replayer threads during replay in the current instance
speed	int8	Log replay rate of the current instance
local_max_ptr	int8	Maximum number of replay logs received by the localhost after the instance is started.
primary_flush_ptr	int8	Position where the host flushes logs to a disk
worker_info	text	Replayer thread information of the instance. If concurrent replay is not enabled, the value is NULL .

- 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 7-86 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	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.

- `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
- `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
- `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 7-87 local_redo_time_count return parameters

Column	Description
thread_name	Thread name.
step1_total	Total duration of step 1. The process of each thread is as follows: • Ultimate RTO: batch redo: obtains a log from a queue. redo manager: obtains a log from a queue. redo worker: obtains a log from a queue. txn manager: reads a log from a queue. txn worker: reads a log from a queue. read worker: reads an Xlog page (overall) from a file. read page worker: obtains a log from a queue. startup: obtains a log from a queue. • Parallel replay: page redo: obtains a log from a queue. startup: reads a log.
step1_count	Number of accumulated execution times of step 1.
step2_total	Total duration of step 2. The process of each thread is as follows: • Ultimate RTO: batch redo: processes logs (overall). redo manager: processes logs (overall). redo worker: processes logs (overall). txn manager: processes logs (overall). txn worker: processes logs (overall). redo worker: specifies the time required for reading the Xlog page. read page worker: generates and sends LSN forwarders. startup: checks whether to replay to the specified position. • Parallel replay: page redo: processes logs (overall). startup: checks whether to replay to the specified position.
step2_count	Number of accumulated execution times of step 2.

Column	Description
step3_total	Total duration of step 3. The process of each thread is as follows: • Ultimate RTO: batch redo: updates the standby state. redo manager: processes data logs. redo worker: replays page logs (overall). txn manager: updates the flushing LSN. txn worker: replays logs. redo worker: updates the Xlog segment. read page worker: obtains a new item. startup: collects statistics on the wait time of delayed replay feature. • Parallel replay: page redo: updates the standby state. startup: collects statistics on the wait time of delayed replay feature.
step3_count	Number of accumulated execution times of step 3.
step4_total	Total duration of step 4. The process of each thread is as follows: • Ultimate RTO: batch redo: parses Xlogs. redo manager: processes DDL operations. redo worker: reads data pages. txn manager: synchronizes the wait time. txn worker: updates the LSN of the current thread. read page worker: stores logs in the distribution thread. startup: distributes logs (overall). • Parallel replay: page redo: replays undo logs. startup: distributes logs (overall).
step4_count	Number of accumulated execution times of step 4.

Column	Description
step5_total	Total duration of step 5. The process of each thread is as follows: • Ultimate RTO: batch redo: distributes logs to the redo manager. redo manager: distributes logs to redo workers. redo worker: replays data page logs. txn manager: distributes data to the txn worker. txn worker: forcibly synchronizes the wait time. read page worker: updates the LSN of the current thread. startup: decodes logs. • Parallel replay: page redo: replays sharetxn logs. startup: replays logs.
step5_count	Number of accumulated execution times of step 5.
step6_total	Total duration of step 6. The process of each thread is as follows: • Ultimate RTO: redo worker: replays non-data page logs. txn manager: updates global LSNs. read page worker: performs CRC on logs. • Parallel replay: page redo: replays synctrxn logs. startup: forcibly synchronizes the wait time.
step6_count	Number of accumulated execution times of step 6.
step7_total	Total duration of step 7. The process of each thread is as follows: • Ultimate RTO: redo manager: creates tablespaces. redo worker: updates FSM. • Parallel replay: page redo: replays a single log.
step7_count	Number of accumulated execution times of step 7.

Column	Description
step8_total	Total duration of step 8. The process of each thread is as follows: • Ultimate RTO: redo worker: forcibly synchronizes the wait time. • Parallel replay: page redo: replays all workers do logs.
step8_count	Number of accumulated execution times of step 8.
step9_total	Total duration of step 9. The process of each thread is as follows: • Ultimate RTO: None • Parallel replay: page redo: replays multi-worker do logs.
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 7-88 `local_xlog_redo_statics` parameters

Column	Description
xlog_type	Log types.
rmid	Resource manager ID.
info	Xlog operation.
num	Number of logs.
extra	Valid values are available for page replay logs and xact logs. • The page replay logs indicate the number of pages read from the disk. • The xact logs indicate the number of deleted files.

- `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 7-89 remote_bgwriter_stat parameter description

Parameter	Type	Description
node_name	text	Instance 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_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 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.)

```
openGauss=# 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
-----+-----+-----+-----+-----+-----
 datanode2 |          0 |          0 |      393208 |          665 |          0
 datanode1 |          0 |          0 |      393208 |          659 |          0
(2 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.

```
openGauss=# 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.

```
openGauss=# 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
```

7.5.25 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
- **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
Return type: text

7.5.26 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:

```
openGauss=# 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:

```
openGauss=# select bucketbool(true,1);
 bucketbool
-----
          1
(1 row)
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# select getbucket(10,'H');
getbucket
-----
       14535
(1 row)

openGauss=# select getbucket(11,'H');
getbucket
-----
       13449
(1 row)

openGauss=# select getbucket(11,'R');
getbucket
-----
       13449
(1 row)

openGauss=# select getbucket(12,'R');
getbucket
-----
        9412
(1 row)
```

- 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:

```
openGauss=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
hash_array
-----
-382888479
(1 row)
```

- **hash_group(key)**

Description: Calculates the hash value of each column in the Group Clause in the streaming engine. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

Parameter: **key** indicates the value of each column in the Group Clause.

Return type: 32-bit hash value

Example:

Perform the following steps in sequence.

```
openGauss=# CREATE TABLE tt(a int, b int,c int,d 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
```

```
openGauss=# select * from tt;
```

```
a | b | c | d
---+---+---+---
(0 rows)
```

```
openGauss=# insert into tt values(1,2,3,4);
```

```
INSERT 0 1
```

```
openGauss=# select * from tt;
```

```
a | b | c | d
---+---+---+---
1 | 2 | 3 | 4
(1 row)
```

```
openGauss=# insert into tt values(5,6,7,8);
```

```
INSERT 0 1
```

```
openGauss=# select * from tt;
```

```
a | b | c | d
---+---+---+---
1 | 2 | 3 | 4
5 | 6 | 7 | 8
(2 rows)
```

```
openGauss=# select hash_group(a,b) from tt where a=1 and b=2;
```

```
hash_group
-----
990882385
(1 row)
```

- **hash_numeric(numeric)**

Description: Calculates the hash value of numeric data.

Parameter: data of the numeric type.

Return type: integer

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# select hashbpchar('hello');
 hashbpchar
-----
-1870292951
(1 row)

openGauss=# 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:

```
openGauss=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
openGauss=# call hashenum('good':b1);
 hashenum
-----
 1821213359
(1 row)
```

- **hashfloat4(real)**

Description: Converts float4 values to hash values.

Parameter: data of the real type

Return type: integer

Example:

```
openGauss=# 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:

```
openGauss=# select hashfloat8(123456.1234);
hashfloat8
-----
1673665593
(1 row)
```

- hashinet(inet)

Description: Supports hashing indexes on inet or cidr. Returns the hash value of inet.

Parameter: data of the inet type

Return type: integer

Example:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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

- **buckettimestampz**
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

7.5.27 Prompt Message Function

- **report_application_error()**
Description: This function can be used to throw errors during PL execution.
Return type: void

Table 7-90 report_application_error parameter description

Parameter	Type	Description	Required (Yes/No)
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

```
openGauss=# CREATE OR REPLACE FUNCTION GET_RESULT_UNKNOWN(  
opengauss(# IN context_id int, /*context_id*/  
opengauss(# IN col_type text /*col_type*/  
opengauss(# )RETURNS INTEGER  
opengauss-# AS $$  
opengauss$# BEGIN  
opengauss$# IF col_type IS NULL THEN  
opengauss$# PG_CATALOG.REPORT_APPLICATION_ERROR('invalid input for the third parameter  
col_type should not be null');  
opengauss$# END IF;  
opengauss$# PG_CATALOG.REPORT_APPLICATION_ERROR('UnSupport data type for  
column_value(context: '|context_id||', '|PG_CATALOG.QUOTE_LITERAL(col_type)||')');  
opengauss$# RETURN -1;  
opengauss$# END;  
opengauss$# $$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION
opengauss=# CALL GET_RESULT_UNKNOWN(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')"
```

```
PL/pgSQL function get_result_unknown(integer,text) line 4 at PERFORM
```

7.5.28 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

7.5.29 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)`
- `pg_enable_redis_proc_cancelable()`
- `pg_disable_redis_proc_cancelable()`
- `pg_tupleid_get_blocknum(tid)`
- `pg_tupleid_get_offset(tid)`
- `pg_tupleid_get_ctid_to_bigint(ctid)`

7.5.30 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 7-91 Parameter description of init

Parameter	Type	Description	Required (Yes/No)
kind	char	Recommendation type. Currently, this parameter can only be set to D .	Yes
isUseCost	boolean	Specifies whether optimizers are used. If data is available, optimizers are used.	Yes
isUseCollect	boolean	Specifies whether the analysis is started from the collected load. The default value is false .	No
isConstraint PrimaryKey	boolean	Specifies whether primary key constraints are retained. The default value is true .	No
sqlCount	int	Number of collected SQL statements. The default value is 10000 . The value ranges from 1 to 100000.	No
maxMemory	int	Maximum memory occupied by distribution key recommendation. The default value is 1024 . 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 called. This function does not need to be called during analysis.

Return type: Boolean

Table 7-92 Parameter description of set_weight_params

Parameter	Type	Description	Required (Yes/No)
joinWeight	real	Weight of JOIN. The value ranges from 0 to 1000.	Yes
groupbyWeight	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 WhatIf cost model.

Return type: Boolean

Table 7-93 Parameter description of `set_cost_params`

Parameter	Type	Description	Required (Yes/No)
<code>maxTime</code>	<code>bigint</code>	Maximum recommendation duration, in minutes. If the value is less than or equal to 0 , the duration is not limited by default.	Yes
<code>isTotalSQL</code>	<code>boolean</code>	Specifies whether all SQL statements are used for calculation. The value true indicates that all SQL statements are used for calculation. The value false indicates that SQL statements whose cost is too high or too low are filtered out based on the percentile.	Yes
<code>compressLevel</code>	<code>text</code>	Search space size of the recommendation algorithm. The options are low , med , and high .	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 called.

- `sqladvisor.analyze_query(text, int)`

Description: Imports SQL statements to be recommended and analyzes the components of the statements.

Return type: Boolean

Table 7-94 Parameter description of analyze_query

Parameter	Type	Description	Required (Yes/No)
query	text	SQL statement	Yes
frequency	int	Frequency of a statement in the load. The default value is 1 . 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
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
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 7-95 Parameter description of start_collect_workload

Parameter	Type	Description	Required (Yes/No)
sqlCount	int	Maximum number of SQL statements for online load collection. The value ranges from 1 to 100000 and the default value is 10000 .	Yes
maxMemory	int	Maximum memory occupied by online load collection. The default value is 1024 . The value ranges from 1 to 10240, in MB.	Yes

NOTICE

- The online collection function can be called only by a 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 a system administrator.

- `sqladvisor.clean_workload()`

Description: Clears the memory in the load.

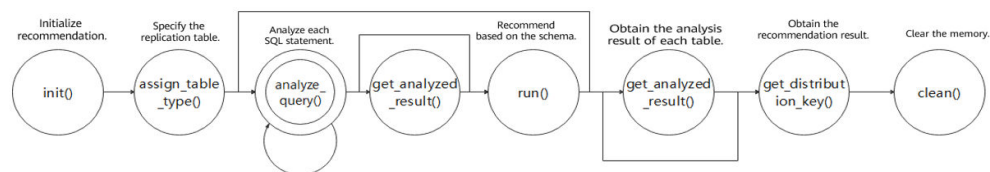
Return type: Boolean

NOTICE

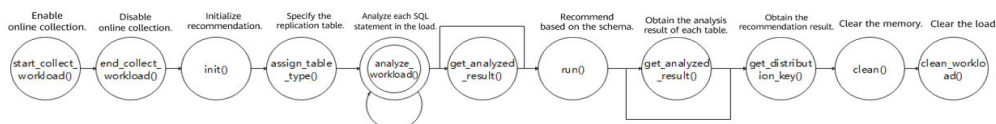
- The function of clearing memory in the load can be called only by a system administrator.
- You must manually execute the cleanup function.

Suggestions

- Call state machines in the heuristic or WhatIf cost recommendation mode.



- Call state machines in semi-online recommendation mode.



7.5.31 Internal Functions

The following functions of GaussDB use internal data types, which cannot be directly called by users.

- Selectivity calculation functions

areajoin el	areasel	arraycon tjoin sel	arraycon tsel	contjoin el	contsel	eqjoin sel
eqsel	iclikejoin sel	iclikesel	icnlikejoi nsel	icnlikese l	icregexe qjoin sel	icregexe qsel
icregexn ejoin sel	icregexn esel	likejoin el	likesel	neqjoin el	neqsel	nlikejoin sel
nlikesel	positionj oin sel	positions el	regexeqj oin sel	regexeqs el	regexnej oin sel	regexnes el
scalargtj oin sel	scalargts el	scalartj oin sel	scalartls el	tsmatchj oin sel	tsmatchs el	-

- Statistics collection functions

array_tyanalyze	range_tyanalyze	ts_tyanalyze
local_rto_stat	remote_rto_stat	-

- Internal functions for sorting

bpchar_sorts upport	bytea_sortsu pport	date_sortsup port	numeric_sort support	timestamp_s ortsupport
------------------------	-----------------------	----------------------	-------------------------	---------------------------

- Internal type processing functions

abstimer ecv	euc_jis_2 004_to_ utf8	int2recv	line_recv	oidvecto rrecv_ext end	tidrecv	utf8_to_ koi8u
-----------------	------------------------------	----------	-----------	------------------------------	---------	-------------------

anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	interval_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv
big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tinterval_recv	varbit_transform
bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform
boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharr_recv
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	gin_extract_vector	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalrecv	gtsvector_compress	json_recv	mic_to_latin4	regoperr_recv	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	regtypercv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimercv	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	large_seq_upgrade_ntree	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	cstore_tid_out
numeric_bool	int2vector_extend	int2vectorout_extend	int2vectorrecv_extend	int2vector_send_extend	int8_accum	large_seq_rollback_ntree
i16toi1	int16eq	int16ge	int16gt	int16in	int16le	int16lt
int16mi	int16mul	int16ne	int16out	int16pl	int16recv	int16send

int16_bo ol	-	-	-	-	-	-
----------------	---	---	---	---	---	---

- Internal functions for aggregation operations

array_ag g_finalfn	array_ag g_transf n	bytea_st ring_agg _finalfn	bytea_st ring_agg _transfn	date_list _agg_no arg2_tra nsfn	date_list _agg_tra nsfn	float4_li st_agg_n oarg2_tr ansfn
float4_li st_agg_t ransfn	float8_li st_agg_n oarg2_tr ansfn	float8_li st_agg_tr ansfn	int2_list _agg_noa rg2_tran sfn	int2_list _agg_tran sfn	int4_list _agg_noa rg2_tran sfn	int4_list _agg_tran sfn
int8_list _agg_noa rg2_tran sfn	int8_list _agg_tran sfn	interval_ list_agg_ noarg2_t ransfn	interval_ list_agg_ transfn	list_agg_f inalfn	list_agg_ noarg2_t ransfn	list_agg_ transfn
median	median_f loat8_fin alfn	median_ interval_f inalfn	median_ transfn	mode_fi nal	numeric_ list_agg_ noarg2_ _transfn	numeric_ list_agg_ _transfn
ordered_ set_trans ition	percentil e_cont_fl oat8_fin al	percentil e_cont_i nterval_f inal	string_a gg_finalf n	string_a gg_trans fn	timesta mp_list_ agg_noa rg2_tran sfn	timesta mp_list_ agg_tran sfn
timesta mptz_list _agg_no arg2_tra nsfn	timesta mptz_lis t_agg_tr ansfn	checksu mtext_a gg_trans fn	json_agg _transfn	json_agg _finalfn	json_obj ect_agg_ transfn	json_obj ect_agg_f inalfn

- Hash internal functions

hashbegi nscan	hashbuil d	hashbuil dempty	hashbul kdelete	hashcost estimate	hashend scan	hashget bitmap
hashgett uple	hashinse rt	hashmar kpos	hashmer ge	hashresc an	hashrest rpos	hashvac uumclea nup
hashvarl ena	jsonb_ha sh	-	-	-	-	-

- 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	ubtrestrpos
ubtvacuumcleanup	-	-	-	-

- PL/pgSQL 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	-	-	-	-	-

- Internal function related to data skew optimization
distributed_count
- Internal functions related to table statistics

pgxc_get_stat_dirty_tables	pgxc_stat_dirty_tables	get_global_stat_all_tables	get_summary_stat_all_tables
----------------------------	------------------------	----------------------------	-----------------------------

- Function for reading data remotely
gs_read_block_from_remote is used to read the pages of a non-segment-page table file. By default, only the initial user can view the data. Other users can view the data only after being granted permissions.
gs_read_segment_block_from_remote is used to read the pages of a segment-page table file. By default, only the initial user can view the data. Other users can view the data only after being granted 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 view the data only after being granted 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 view the data only after being granted permissions.
- View-related reference functions
adm_hist_sqlstat_func
adm_hist_sqlstat_idlog_func

7.5.32 Dynamic Data Masking Functions

NOTE

This function is an internal function. For details, see "Database Security > Dynamic Data Anonymization" in *Feature Description*.

- creditcardmasking(col text, letter char default 'x')
Description: Replaces the digits before the last four bits following the col string with letters.
Parameter: Character string to be replaced or character string used for replacement
Return type: text
- basicmailmasking(col text, letter char default 'x')
Description: Replaces the characters before the first at sign (@) in the col string with letters.
Parameter: Character string to be replaced or character string used for replacement
Return type: text
- fullmailmasking(col text, letter char default 'x')
Description: Replaces the characters (except @) before the last period (.) in the col string with letters.
Parameter: Character string to be replaced or character string used for replacement
Return type: text

- `alldigitsmasking(col text, letter char default '0')`
Description: Replaces the digits in the col string with letters.
Parameter: Character string to be replaced or character string used for replacement
Return type: text
- `shufflemasking(col text)`
Description: Sorts the characters in the col string out of order.
Parameter: Character string to be replaced or character string used for replacement
Return type: text
- `randommasking(col text)`
Description: Randomizes the characters in the col string.
Parameter: Character string to be replaced or character string used for replacement
Return type: text
- `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: Character string to be replaced, regular expression, replacement start position, and replacement length.
Return type: text

7.5.33 Hotkey Feature Functions

- `gs_stat_get_hotkeys_info()`
Description: Obtains hotkey information queried on the local node.
Return type: Tuple
Example:

```
openGauss=# 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)
```
- `gs_stat_clean_hotkeys()`
Description: Clears hotkey cache and resets hotkey status information.
Return type: bool, and the return value is always **true**.

Example:

```
openGauss=# select * from gs_stat_clean_hotkeys();
gs_stat_clean_hotkeys
-----
t
(1 row)
```

7.5.34 Global SysCache Feature Functions

- `gs_gsc_table_detail(database_id default NULL, rel_id default NULL)`

Description: Queries global system cache table metadata in a database. The user who calls this function must have the SYSADMIN permission.

Parameter: Specifies the database and table to be queried in the global system cache. The default value **NULL** or value **-1** of **database_id** indicates all databases. The value **0** indicates a shared table. Other values indicate a specified database and a shared table. **rel_id** indicates the OID of a specified table. The default value **NULL** or value **-1** indicates all tables. Other values indicate a specified table. If **database_id** does not exist, an error is reported. If **rel_id** does not exist, the query result is empty.

Return type: Tuple

```
select * from gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid | relname | relnamespace | reltype | reloftype |
relowner | relam | relfilenode | reltablespace | relhasindex | relisshared | relkind | relnatts | relhasoids |
relhaspkey | parttype | tdhasuids | attnames | extinfo
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
0 |  | 2676 | pg_authid_rolname_index | 11 | 0 | 0 | 10 | 403 | 0
| 1664 | f | t | i | 1 | f | f | n | f | 'rolname' |
(1 row)
```

- **gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)**

Description: Queries the system catalog row information in the global system cache in a database. The user who calls this function must have the SYSADMIN permission.

Parameter: Specifies the database and table to be queried in the global system cache. The default value **NULL** or value **-1** of **database_id** indicates all databases. The value **0** indicates a shared table. Other values indicate a specified database and a shared table. **rel_id** indicates the ID of a specified table, including all system catalogs in the system cache. The default value **NULL** or value **-1** indicates all tables. Other values indicate a specified table. If **database_id** does not exist, an error is reported. If **rel_id** does not exist, the result is empty.

Return type: Tuple

Example:

```
openGauss=#
select * from gs_gsc_catalog_detail(16574, 1260);
database_id | database_name | rel_id | rel_name | cache_id | self | ctid | infomask | infomask2 |
hash_value | refcount
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
10 | 0 | 1260 | pg_authid | 10 | (0, 9) | (0, 9) | 10507 | 26 | 531311568 |
0 | 0 | 1260 | pg_authid | 11 | (0, 4) | (0, 4) | 2313 | 26 | 365368336 | 1
0 | 0 | 1260 | pg_authid | 11 | (0, 9) | (0, 9) | 10507 | 26 | 3911517328 |
10 | 0 | 1260 | pg_authid | 11 | (0, 7) | (0, 7) | 2313 | 26 | 1317799983 |
1 | 0 | 1260 | pg_authid | 11 | (0, 5) | (0, 5) | 2313 | 26 | 3664347448 |
1 | 0 | 1260 | pg_authid | 11 | (0, 1) | (0, 1) | 2313 | 26 | 276477273 | 1
0 | 0 | 1260 | pg_authid | 11 | (0, 3) | (0, 3) | 2313 | 26 | 2465837659 |
1 | 0 | 1260 | pg_authid | 11 | (0, 8) | (0, 8) | 2313 | 26 | 3205288035 |
1 | 0 | 1260 | pg_authid | 11 | (0, 6) | (0, 6) | 2313 | 26 | 131811687 | 1
0 | 0 | 1260 | pg_authid | 11 | (0, 2) | (0, 2) | 2313 | 26 | 1226484587 |
(10 rows)
```

- `gs_gsc_clean(database_id default NULL)`

Description: Clears the global syscache cache. Note that data in use will not be cleared. The user who calls this function must have the SYSADMIN permission.

Parameter: Specifies the database whose global system cache needs to be cleared. The default value **NULL** or **-1** indicates that the global system cache of all databases is cleared. The value **0** indicates that the global system cache of only the shared table is cleared. Other values indicate that the global system cache of the specified database and shared table is cleared. If **database_id** does not exist, an error is reported.

Return type: Boolean

Example:

```
openGauss=# select * from gs_gsc_clean();
gs_gsc_clean
-----
t
(1 row)
```

- `gs_gsc_dbstat_info(database_id default NULL)`

Description: Obtains GSC memory statistics on the local node, including cache query, hit, loading, expiration, and occupied space information of tuples, relationships, and partitions, database-level elimination information, thread reference information, and memory usage information. This parameter can be used to locate performance problems. For example, if the value of the hits/searches array is far less than 1, the value of **global_syscache_threshold** may be too small. As a result, the query hit ratio decreases. The user who calls this function must have the SYSADMIN permission.

Parameter: Specifies the global system cache statistics of the database to be queried. **NULL** or **-1** indicates that all databases are queried. **0** indicates that only the information about the shared table is queried. Other values indicate that the information about the specified database and shared table is queried. If an invalid value is entered, an error is reported, indicating that **database_id** does not exist.

Return type: Tuple

Example:

```

openGauss=# select * from gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_dead | tup_memory
| rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_count | part_dead |
part_memory | total_memory | swa
pout_count | refcount
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      0 |      |      300 |    235 |    31 |    22 |    2 |    9752 |      598 |    108
|      1
8 |      18 |    0 |    77720 |      0 |    0 |    0 |    0 |    0 |    0 |    752912 |
|      0 |    0
16574 | testdb |      3368 |    2289 |    329 |    273 |    0 |    92593 |      1113 |
524 |      4
8 |      48 |    0 |    340456 |      0 |    0 |    0 |    0 |    0 |    0 |    4124792 |
|      0 |    10
(2 rows)

```

7.5.35 Data Damage Detection and Repair Functions

- `gs_verify_data_file(verify_segment bool)`

Description: Checks whether files in the current database of the current instance are lost. Only whether intermediate segments are lost in the main file of the data table is checked. The default value is **false**, indicating that segment-page table data files are not checked. If this parameter is set to **true**, only segment-page table files are checked. By default, only the initial user, users with the SYSADMIN permission, and users with the OPRADMIN permission in the O&M mode can view the information. Other users can view the information only after being granted with permissions.

The returned result is as follows:

- Non-segment-page table: **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.
- Segment-page table: All tables are stored in the same file. Therefore, **rel_oid** and **rel_name** cannot display information about a specific table. For a segment-page table, if the first file is damaged, the subsequent files such as .1 and .2 are not checked. For example, if files 3, 3.1, and 3.2 are damaged, only damage of file 3 can be detected. When the number of segment-page files is less than 5, the files that are not generated are also checked during function detection. For example, if there are only files 1 and 2, files 3, 4, and 5 are checked during segment-page file detection. In the following examples, the first is an example of checking a non-segment-page table, and the second is an example of checking a segment-page table.

Parameter description:

- `verify_segment`

Specifies the range of files to be checked. **false** indicates that non-segment-page tables are checked. **true** indicates that segment-page tables are checked.

The value can be **true** or **false** (default value).

Return type: record.

Example (The abnormal line is displayed only when an exception is detected. Otherwise, no line is displayed.):

Check a non-segment-page table.

```
openGauss=# select * from gs_verify_data_file();
node_name      | rel_oid | rel_name      | miss_file_path
-----+-----+-----+-----
dn_6001_6002_6003 | 16554 | test         | base/16552/24745
```

Check a segment-page table.

```
openGauss=# select * from gs_verify_data_file(true);
node_name      | rel_oid | rel_name      | miss_file_path
-----+-----+-----+-----
dn_6001_6002_6003 | 0 | none         | base/16573/2
```

- `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. The value of table OID for a segment-page table ranges from 0 to

4294967295. (The internal verification determines whether a file is a segment-page table file based on the file path. The table OID is not used for a segment-page table file.) 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 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 with 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.

Parameter description:

- **tableoid**
Specifies the 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**
Specifies the 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 **tableoid** and **path** based on the output of **gs_verify_data_file**):

```
openGauss=# 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 with permissions. **file_path** indicates the relative path of the damaged file. If the table is a segment-page table, the logical information instead of the actual physical file information is displayed. **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.):

```
openGauss=# 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 with 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 monitoring users can view the information. Other users can view the information only after being granted with permissions.

Return type: Boolean.

Example:

```
openGauss=# 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 monitoring users can view the information. Other users can view the information only after being granted with 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 with 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:

1. 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.
2. 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 disks.

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**. If you want to verify the undo pages of the Ustore table, enter the block number 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 those 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 automatic repair function through remote read 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. Set this parameter based on the value of **bucket_node** in **local_bad_block_info**. If the value of **bucket_node** is **-1**, the table is not a segment-page table. In this case, set **is_segment** to **false**. If the value of **bucket_node** is not **-1**, set **is_segment** to **true**.
Value range: The value is of a Boolean type and can be **true** or **false**.

Return type: record.

Examples (Set parameters based on the output of `local_bad_block_info`. Otherwise, an error is reported.):

```
openGauss=# 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: Restores the specified page of the instance. This function can be used only by the primary DN that is properly connected to the primary and standby DNs. If the page is successfully restored, **true** is returned. If an error occurs during the restoration, an error message is displayed. 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 with 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.

Parameters:

- `path`
Specifies the path of the damaged page. Set this parameter based on the **file_path** column in **local_bad_block_info** or the **path** column in **gs_verify_and_tryrepair_page**.
Value range: a string
- `blocknum`
Specifies the number of the damaged page. Set this parameter based on the **block_num** column in **local_bad_block_info** or the **blocknum** column in **gs_verify_and_tryrepair_page**.
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. The value of this parameter is determined by the value of **bucket_node** in **local_bad_block_info**. If the value of **bucket_node** is **-1**, the table is not a segment-page table and **is_segment** is set to **false**. If the value of **bucket_node** is not **-1**, **is_segment** is set to **true**.
Value range: The value is of 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 replayed 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 (Set parameters based on the output of `local_bad_block_info`. Otherwise, an error is reported.):

```
openGauss=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
```

t

7.5.36 Other System Functions

Other system functions are classified into PostgreSQL-compatible functions and internal functions. These functions are not recommended. If you need to use them, contact Huawei technical support.

PostgreSQL-compatible Functions and Operators

The following table lists the built-in functions and operators of GaussDB that are compatible with PostgreSQL.

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expandarray	_pg_index_position	_pg_interval_type	_pg_numeric_precision
_pg_numeric_precision_radix	_pg_numeric_scale	_pg_typeof	_pg_typeof	q	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimeelt	abstimene
abstimeout	abstimerectv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
aclitemeq	aclitemin	aclitemout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_recv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinse	areasele	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_le	array_length	array_lower	array_lt	array_ndims
array_ne	array_out	array_prepend	array_rectv	array_send	array_smaller	array_to_json
array_to_string	array_type_analyze	array_upper	arraycontains	arraycontains	arraycontjoin	arraycontsel

arrayoverla p	ascii	asin	atan	atan2	avals	avg
big5_to_euc _tw	big5_to_ mic	big5_to_u tf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_sen d	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot
bitor	bitshiftle ft	bitshiftrig ht	bittyp modin	bittypm odout	bitxor	bool
bool_and	bool_or	booland_ statefunc	booleq	boolge	boolgt	boolin
boolle	boollt	boolne	boolor_ statef unc	boolout	boolre cv	boolsend
box	box_abov e	box_abov e_eq	box_ad d	box_bel ow	box_b elow_e q	box_center
box_contain	box_cont ain_pt	box_cont ained	box_dis tance	box_div	box_e q	box_ge
box_gt	box_in	box_inter sect	box_le	box_left	box_lt	box_mul
box_out	box_over above	box_over below	box_ov erlap	box_ove rleft	box_ov erright	box_recv
box_right	box_sam e	box_send	box_su b	bpchar	bpchar_ larger	bpchar_patt ern_ge
bpchar_patt ern_gt	bpchar_p attern_le	bpchar_p attern_lt	bpchar_ small er	bpchar_ sortsup port	bpchar cmp	bpchareq
bpcharge	bpchargt	bpcharicli ke	bpchari cnlike	bpcharic regexeq	bpchar icrege xne	bpcharin
bpcharle	bpcharlik e	bpcharlt	bpchar ne	bpcharn like	bpchar out	bpcharrecv
bpcharregex eq	bpcharre gexne	bpcharse nd	bpchar typmo din	bpchart ypmodo ut	broadc ast	btabstimec mp
btdarraycmp	btbegins can	btboolcm p	btbpch ar_patt ern_cm p	btbuild	btbuil dempt y	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
btreststrpos	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	cbirt	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_overright	circle_recv
circle_right	circle_same	circle_send	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	currval	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_timestamptz	date_ge	date_ge_timestamp	date_ge_timestamptz
date_gt	date_gt_timestamp	date_gt_timestamptz	date_in	date_larger	date_le	date_le_timestamp
date_le_timestamptz	date_lt	date_lt_timestamp	date_lt_timestamptz	date_mi	date_mi_interval	date_mii
date_ne	date_ne_timestamp	date_ne_timestamptz	date_out	date_pl_interval	date_pli	date_recv

date_send	date_smaller	date_sortsupport	daterange_canonical	daterange_subdiff	datetime_pl	datetimetz_pl
dcbt	decode	defined	degrees	delete	dense_rank	dexp
diagonal	diameter	dispell_init	dispell_lexize	dist_cpoly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppath	dist_ps	dist_sb	dist_sl	div
dlog1	dlog10	domain_in	domain_recv	dpow	dround	dsimple_init
dsimple_lexize	dsnowball_init	dsnowball_lexize	dsqrt	dsynonym_init	dsynonym_lexize	dtrunc
each	enum_ne	enum_out	enum_range	enum_recv	enum_send	enum_smaller
eqjoinsel	eqsel	euc_cn_to_mic	euc_cn_to_utf8	euc_jis_2004_to_shift_jis_2004	euc_jis_2004_to_utf8	euc_jp_to_mic
euc_jp_to_sjis	euc_jp_to_utf8	euc_kr_to_mic	euc_kr_to_utf8	euc_tw_to_big5	euc_tw_to_mic	euc_tw_to_utf8
every	exist	exists_all	exists_any	exp	factorial	family
fdw_handler_in	fdw_handler_out	fetchval	first_value	float4	float4_accum	float48div
float48eq	float48ge	float48gt	float48le	float48lt	float48mi	float48mul
float48ne	float48pl	float4abs	float4div	float4eq	float4ge	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_intcept	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	float8abs	float8div	float8eq	float8ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul
float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_c_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_large_table_name	gin_clean_pending_list	gin_cmp_prefix	gin_cmp_tsl_exeme
gin_extract_tsquery	gin_extract_tsvector	gin_tsquery_consistent	gin_tsquery_triconsistent	ginarray_consistent	ginarrayextract	ginarraytriconsistent
ginbeginscan	ginbuild	ginbuildempty	ginbulkdelete	gincostestimate	ginendscan	gingetbitmap
gininsert	ginmarkpos	ginoptions	ginqueryarrayextract	ginrescan	ginrestpos	ginvacuumcleanup
gist_box_compress	gist_box_consistent	gist_box_decompress	gist_box_penalty	gist_box_picksplit	gist_box_same	gist_box_union
gist_circle_compress	gist_circle_consistent	gist_point_compressed	gist_point_consistent	gist_point_distance	gist_poly_compressed	gist_poly_consistent
gistbeginscan	gistbuild	gistbuildempty	gistbulkdelete	gistcostestimate	gisten_scan	gistgetbitmap

gistgettupl	gistinsert	gistmark pos	gistopty ons	gistresc an	gistres trpos	gistvacuumc leanup
gtsquery_co mpress	gtsquery _consiste nt	gtsquery_ decompr ess	gtsque ry_pen alty	gtsquer y_picksp lit	gtsque ry_sa me	gtsquery_un ion
gtsvector_co mpress	gtsvector _consiste nt	gtsvector _decompr ess	gtsvect or_pen alty	gtsvecto r_picksp lit	gtsvec tor_sa me	gtsvector_u nion
gtsvectorin	gtsvector out	has_table space_pri vilege	has_ty pe_priv ilege	hash_ac litem	hashb eginsc an	hashbuild
hashbuilde mpty	hashbulk delete	hashcost estimate	hashen dscan	hashget bitmap	hashg ettupl e	hashinsert
hashint2vec tor	hashint4	hashint8	hashm acaddr	hashma rkpos	hashn ame	hashoid
hashoidvect or	hashopti ons	hashresca n	hashre strpos	hashtex t	hashv acuun cleanu p	hashvarlena
host	hostmas k	iclikejoins el	iclikese l	icnlikejo insel	icnlike sel	icregexeqjoi nsel
icregexeqsel	icregexne joinsel	icregexne sel	inet_cli ent_ad dr	inet_clie nt_port	inet_in	inet_out
inet_rcv	inet_sen d	inet_serv er_addr	inet_se rver_po rt	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_acc um	int2_a vg_acc um	int2_mul_ca sh
int2_sum	int24div	int24eq	int24g e	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28e q	int28ge
int28gt	int28le	int28lt	int28m i	int28mu l	int28n e	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2m ul	int2ne

int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vectorreq	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	int8um	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_h	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_timestamp_tz
interval_pl_timestamp	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_distanc e	lseg_eq	lseg_ge	lseg_gt	lseg_hor izontal	lseg_in	lseg_interpt
lseg_interse ct	lseg_le	lseg_leng th	lseg_lt	lseg_ne	lseg_o ut	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_ve rtical	ltrim	macad dr_and	macaddr_c mp
macaddr_eq	macaddr _ge	macaddr_ gt	macad dr_in	macadd r_le	macad dr_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	nlike se 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_col lect	numeric_ div	numeric_ div_trunc	numeri c_eq	numeric _exp	numeri c_fac	numeric_ge
numeric_gt	numeric_ in	numeric_ inc	numeri c_large r	numeric _le	numeri c_ln	numeric_log
numeric_lt	numeric_ mod	numeric_ mul	numeri c_ne	numeric _out	numeri c_pow er	numeric_rec v
numeric_se nd	numeric_ smaller	numeric_ sortsuppo rt	numeri c_sqrt	numeric _stddev _pop	numeri c_stddev_ sam p	numeric_sub
numeric_tra nsform	numeric_ uminus	numeric_ uplus	numeri c_var_p op	numeric _var_sam p	numeri c_tpm odin	numeri c_tpm modout
numrange_s ubdiff	oid	oid eq	oid ge	oid gt	oid in	oid larger
oidle	oidlt	oidne	oidout	oidrecv	oidsen d	oidsmaller
oidvector eq	oidvector ge	oidvector gt	oidvect orin	oidvect orle	oidvec torlt	oidvectorne
oidvectorou t	oidvector recv	oidvector send	oidvect ortypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque _in	opaque _out	ordere d_set_t ransiti on	overlaps
overlay	path	path_add	path_a dd_pt	path_ce nter	path_c ontain _pt	path_distan ce
path_div_pt	path_in	path_inte r	path_le ngth	path_m ul_pt	path_n _eq	path_n_ge

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_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_statistics_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_dbb_blk_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_start_up_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_recv	point_right	point_send	point_sub	point_vert	poly_above
poly_below	poly_center	poly_contains	poly_contains_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_recv	poly_right	poly_same	poly_send	polygon	position
positionjoin sel	positions el	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_xmlschema
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_gist_compress	range_gist_consistent	range_gist_decompress	range_gist_penalty	range_gist_picksplit
range_gist_same	range_gist_union	range_gt	range_in	range_intersect	range_le	range_lt
range_minus	range_ne	range_out	range_overlaps	range_overleft	range_overright	range_recv
range_send	range_type_analyze	range_union	rank	record_eq	record_ge	record_gt
record_in	record_le	record_lt	record_ne	record_out	record_recv	record_send
regclass	regclassin	regclassout	regclassrecv	regclasssend	regconfigin	regconfigout
regconfigrecv	regconfigsend	regdictionaryin	regdictionaryout	regdictionaryrecv	regdictionarysend	regexeqjoinsel
regexeqsel	regexnejoinsel	regexnesel	regexp_matches	regexp_replace	regexp_split_to_array	regexp_split_to_table
regoperatorin	regoperatorout	regoperatorrecv	regoperatorsend	regoperatorin	regoperatorout	regoperatorrecv
regopersend	regprocedurein	regprocedureout	regprocedurerecv	regproceduresend	regprocin	regprocout
regprocrecv	regprocsend	regr_avgx	regr_avgy	regr_coint	regr_intercept	regr_r2
regr_slope	regr_sxx	regr_sxy	regr_syy	regtypen	regtypeout	regtyperecv

regtypesend	reltime	reltimeeq	reltimege	reltimegt	reltimein	reltimele
reltimelt	reltimein	reltimeout	reltimerecv	reltimesend	repeat	replace
reverse	RI_FKey_cascade_del	RI_FKey_cascade_upd	RI_FKey_check_ins	RI_FKey_check_upd	RI_FKey_noaction_del	RI_FKey_noaction_upd
RI_FKey_restrict_del	RI_FKey_restrict_upd	RI_FKey_setdefault_del	RI_FKey_setdefault_upd	RI_FKey_setnull_del	RI_FKey_setnull_upd	right
round	row_number	row_to_json	rpad	rtrim	scalar_gtjoinsel	scalargtsel
scalarltjoinsel	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	texticnlike	texticregexe	texticregexne
textin	textlike	textne	textnlike	textout	textrecv	textregexe
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_cmp_date	timestamp_cmp_timestamptz
timestamp_eq	timestamp_eq_date	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestamptz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date

timestamp_l e_timestamp ptz	timesta mp_lt	timestamp p_lt_date	timesta mp_lt_t imesta mptz	timesta mp_mi	timestamp_ mi_int erval	timestamp_ ne
timestamp_ ne_date	timesta mp_ne_t imestamp tz	timestamp p_out	timesta mp_pl_ interval	timesta mp_recv	timestamp_ send	timestamp_ smaller
timestamp_ sortsupport	timesta mp_trans form	timestamp typmodi n	timesta mptyp modou t	timesta mptz	timestamp_ cmp	timestamptz _cmp_date
timestamp z_cmp_time stamp	timesta mptz_eq	timestamp tz_eq_d ate	timesta mptz_e q_time stamp	timesta mptz_g e	timestamp_ ge_da te	timestamptz _ge_timesta mp
timestamp z_gt	timesta mptz_gt_ date	timestamp tz_gt_t imestamp	timesta mptz_i n	timesta mptz_la rger	timestamp_ le	timestamptz _le_date
timestamp z_le_timesta mp	timesta mptz_lt	timestamp tz_lt_da te	timesta mptz_lt_ timest amp	timesta mptz_m i	timestamp_ mi_in terval	timestamptz _ne
timestamp z_ne_date	timesta mptz_ne_ timesta mp	timestamp tz_out	timesta mptz_p l_interv al	timesta mptz_re cv	timestamp_ send	timestamptz _smaller
timestamp ztypmodin	timesta mptztyp modout	timetypm odin	timety pmodou t	timetz	timetz_ cmp	timetz_eq
timetz_ge	timetz_gt	timetz_ha sh	timetz_ in	timetz_l arger	timetz_ le	timetz_lt
timetz_mi_i nterval	timetz_n e	timetz_o ut	timetz_ pl_inte rval	timetz_r ecv	timetz_ send	timetz_smal ler
timetzdate_ pl	timetzty pmodin	timetztyp modout	timezo ne (2069)	timezon e (1159)	timezo ne (2037)	timezone (2070)
timezone (1026)	timezone (2038)	tintervalc t	tinterv aleq	tinterval ge	tinterv algt	tintervalin
tintervalle	tintervall eneq	tintervall enge	tinterv allengt	tinterval lenle	tinterv allenlt	tintervallen ne

tintervallt	tinterval ne	tintervalo ut	tinterv alov	tinterval recv	tinterv alsam e	tintervalsen d
tintervalstar t	to_ascii (1845)	to_ascii (1847)	to_ascii (1846)	trigger_i n	trigger _out	ts_match_qv
ts_match_tq	ts_match _tt	ts_match _vq	ts_rank	ts_rank_ cd	ts_rew rite	ts_stat
ts_token_ty pe	ts_typan alyze	tsmatchjo insel	tsmatc hsel	tsq_mco ntained	tsq_m contai ns	tsquery_and
tsquery_cm p	tsquery_ eq	tsquery_g e	tsquery _gt	tsquery_ le	tsquer y_lt	tsquery_ne
tsquery_not	tsquery_ or	tsqueryin	tsquery out	tsqueryr ecv	tsquer ysend	tsrange
tsrange_sub diff	tstzrange	tstzrange _subdiff	tsvecto r_cmp	tsvector _concat	tsvect or_eq	tsvector_ge
tsvector_gt	tsvector_ le	tsvector_l t	tsvecto r_ne	tsvector _update _trigger	tsvect or_upd ate_tri gger_c olumn	tsvectorin
tsvectorout	tsvectorr ecv	tsvectors end	txid_cu rrent	txid_cur rent_sn apshot	txid_sn apshot _in	txid_snapsh ot_out
txid_snapsh ot_recv	txid_snap shot_sen d	txid_snap shot_xip	txid_sn apshot _xmax	txid_sna pshot_x min	txid_vi sible_i n_sna pshot	uhc_to_utf8
unique_key_ recheck	unknown in	unknown out	unkno wnrecv	unknow nsend	unnest	utf8_to_big5
utf8_to_euc _cn	utf8_to_e uc_jis_20 04	utf8_to_e uc_jp	utf8_to _euc_kr	utf8_to_ euc_tw	utf8_t o_gb1 8030	utf8_to_gbk
utf8_to_iso8 859	utf8_to_i so8859_1	utf8_to_j ohab	utf8_to _koi8r	utf8_to_ koi8u	utf8_t o_shift _jis_20 04	utf8_to_sjis
utf8_to_uhc	utf8_to_ win	uuid_cmp	uuid_e q	uuid_ge	uuid_g t	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_n e	uuid_ou t	uuid_r ecv	uuid_send

var_pop	var_samp	varbit	varbit_in	varbit_out	varbit_recv	varbit_send
varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	vachar	vachar_transform	vacharin	vacharout
vacharrecv	vacharsend	vachartypmodin	vachartypmodout	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_wellformed	xml_is_wellformed_content	xml_is_wellformed_document	xml_out	xml_recv	xml_send	xmlagg
xmlcomment	xmlconcat2	xmlexists	xmlvalidate	-	-	-

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.

- smgreq(a smgr, b smgr)**
 Description: Compares two smgrs to check whether they are the same.
 Parameters: smgr, smgr
 Return type: Boolean
- smgrne(a smgr, b smgr)**
 Description: Checks whether the two smgrs are different.
 Parameters: smgr, smgr
 Return type: Boolean
- spread_collect**
 Description: Calculates the difference between the maximum and minimum values within a specified time period. This function is used during the data collection process of aggregate functions.

Parameters: s real[], v real[]

Return type: real[]

- spread_final

Description: Calculates the difference between the maximum and minimum values within a specified time period. This function is used during the final data processing of aggregate functions.

Parameter: s real[]

Return type: real

- spread_internal

Description: Calculates the difference between the maximum and minimum values within a specified time period. This function is used during the intermediate data 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: aggregate function of the list type

Parameters: smallint, text

Return type: text

- log_fdw_validator

Description: validate function

Parameter: text[], oid

Return type: void

- nvarchar2typmodin

Description: Obtains the typmod information of varchar.

Parameter: cstring[]

Return type: integer

- nvarchar2typmodout

Description: Obtains the typmod information of varchar, 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: 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.

Parameter: bigint, "char", bigint, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, smallint, text

Return type: void

- enum_cmp

Description: Enumeration comparison function, which is used to determine whether two enumeration classes are equal and determine their relative sizes.

Parameter: anyenum, anyenum

Return type: integer

- enum_eq

Description: Enumeration comparison function, which is used to implement the = symbol.

Parameter: anyenum, anyenum

Return type: Boolean

- enum_first

Description: Returns the first element in the enumeration class.

Parameter: anyenum

Return type: anyenum

- enum_ge

Description: Enumeration comparison function, which is used to implement the >= symbol.

Parameter: anyenum, anyenum

Return type: Boolean

- enum_gt

Description: Enumeration comparison function, which is used to implement the > sign.

Parameter: anyenum, anyenum

Return type: Boolean

- enum_in

Description: Enumeration comparison function, which is used to determine whether an element is in an enumeration class.

Parameter: cstring, oid

Return type: anyenum

- enum_larger

Description: Enumeration comparison function, which is used to implement the > sign.

Parameter: anyenum, anyenum

Return type: anyenum

- enum_last

Description: Returns the last element in the enumeration class.

Parameter: anyenum

Return type: anyenum

- enum_le

Description: Enumeration comparison function, which is used to implement the <= symbol.

Parameter: anyenum, anyenum

Return type: Boolean

- `enum_lt`
Description: Enumeration comparison function, which is used to implement the < symbol.
Parameter: anyenum, anyenum
Return type: Boolean
- `enum_smaller`
Description: Enumeration comparison function, which is used to implement the < symbol.
Parameter: 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, and 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 xlong 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 7-96 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.

Name	Type	Description
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. • 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.
streamid	OUT bigint	Stream ID in the physical connection.

- **psortoptions**
Description: Returns the psort attribute.
Parameter: 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.
Parameter: xid32, xid32
Return type: Boolean
- **xideqint8**
Description: Compares two xids to check whether they are the same.
Parameter: xid, bigint
Return type: Boolean
- **xidlt**

Description: Returns whether $xid1 < xid2$ is true.

Parameter: xid, xid

Return type: Boolean

- xidlt4

Description: Returns whether $xid1 < xid2$ is true.

Parameter: 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 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 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 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, and **operate_mode** must be enabled for the O&M administrator role.

- gs_set_standby_cluster_target_barrier_id

Description: Sets the target barrier to be played back.

Parameter: barrier_id

Return type: target_id text

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission, and **operate_mode** must be enabled for the O&M administrator role.

- 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, and **operate_mode** must be enabled for the O&M administrator role.

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

7.5.37 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_by_time	dbperf.global_slow_query_history	pg_reload_conf	pg_rotate_logfile	gs_stat_store
pgxc_get_searchlet_info()	pgxc_get_searchlet_table_attr_info()	gs_comm_proxy_thread_status()	-	-	-	-

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.

Examples

```
openGauss=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
result
```

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

openGauss=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
result
-----
t
(1 row)

openGauss=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
result
-----
f
(1 row)

openGauss=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
result
-----
f
(1 row)

openGauss=# SELECT 2+2 IS NULL AS RESULT;
result
-----
f
(1 row)

openGauss=# SELECT 2+2 IS NOT NULL AS RESULT;
result
-----
t
(1 row)

openGauss=# SELECT 2+2 ISNULL AS RESULT;
result
-----
f
(1 row)

openGauss=# SELECT 2+2 NOTNULL AS RESULT;
result
-----
t
(1 row)

openGauss=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
result
-----
t
(1 row)

openGauss=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
result
-----
f
(1 row)
```

7.6.2 Condition Expressions

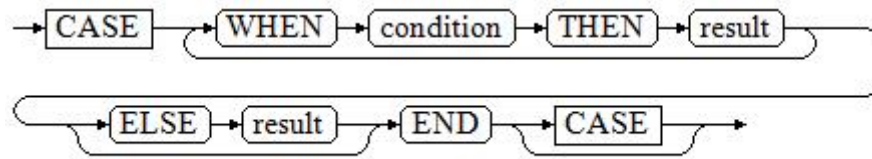
Data that meets the requirements specified by conditional expressions are filtered during SQL statement execution.

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.

For example:

```
openGauss=# CREATE TABLE case_when_t1(CW_COL1 INT);

openGauss=# INSERT INTO case_when_t1 VALUES (1), (2), (3);

openGauss=# SELECT * FROM case_when_t1;
cw_col1
-----
1
2
3
(3 rows)

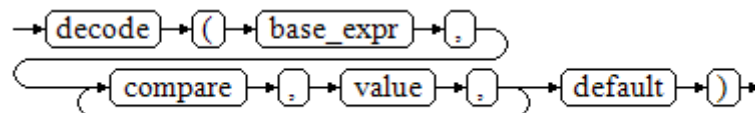
openGauss=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN
'two' ELSE 'other' END FROM case_when_t1 ORDER BY 1;
cw_col1 | case
-----+-----
1 | one
2 | two
3 | other
(3 rows)

openGauss=# 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 no match occurs, default is returned.

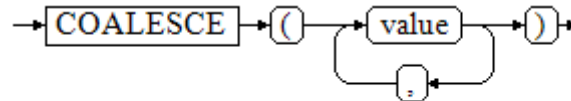
Conditional Expression Functions describes the examples.

```
openGauss=# 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**. It is often used to replace **NULL** with the default value. 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.

Example

```
openGauss=# CREATE TABLE c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10)) ;
```

```
openGauss=# INSERT INTO c_tabl VALUES('abc', 'efg', '123');
openGauss=# INSERT INTO c_tabl VALUES(NULL, 'efg', '123');
```

```
openGauss=# INSERT INTO c_tabl VALUES(NULL, NULL, '123');
```

```
openGauss=# 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)

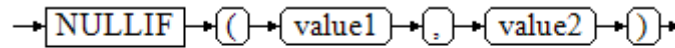
```
openGauss=# DROP TABLE c_tabl;
```

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.

```
openGauss=# 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.

Example

```
openGauss=# CREATE TABLE null_if_t1 (  
    NI_VALUE1 VARCHAR(10),  
    NI_VALUE2 VARCHAR(10)  
);  
  
openGauss=# INSERT INTO null_if_t1 VALUES('abc', 'abc');  
openGauss=# INSERT INTO null_if_t1 VALUES('abc', 'efg');  
  
openGauss=# 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)

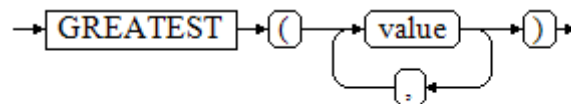
```
openGauss=# DROP TABLE null_if_t1;
```

If **value1** is equal to **value2**, **NULL** is returned. Otherwise, **value1** is returned.

```
openGauss=# SELECT NULLIF('Hello','Hello World');  
nullif  
-----  
Hello  
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

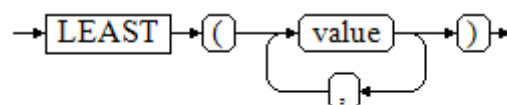
Figure 7-5 shows the syntax of a **GREATEST** expression.

Figure 7-5 greatest::=

You can select the maximum value from any numerical expression list.

```
openGauss=# SELECT greatest(9000,155555,2.01);  
greatest  
-----  
155555  
(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**.

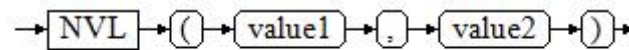
```
openGauss=# SELECT least(9000,2);
least
-----
      2
(1 row)
```

[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.

For example:

```
openGauss=# SELECT nvl(null,1);
nvl
-----
      1
(1 row)

openGauss=# SELECT nvl ('Hello World',1);
nvl
-----
Hello World
(1 row)
```

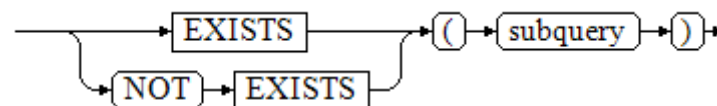
7.6.3 Subquery Expressions

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, or 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.

Example:

```
openGauss=# CREATE TABLE exists_t1(a int, b int);
openGauss=# INSERT INTO exists_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE exists_t2(a int, c int);
openGauss=# INSERT INTO exists_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

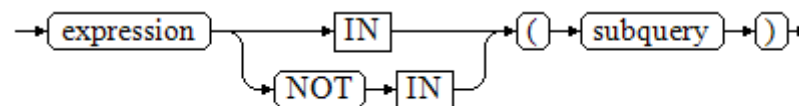
openGauss=# SELECT * FROM exists_t1 t1 WHERE EXISTS (SELECT * FROM exists_t2 t2 WHERE t2.a =
a | b
----+----
3 | 4
4 | 5
(2 rows)

openGauss=# 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.

Example:

```
openGauss=# CREATE TABLE in_t1(a int, b int);
openGauss=# INSERT INTO in_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE in_t2(a int, c int);
openGauss=# INSERT INTO in_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

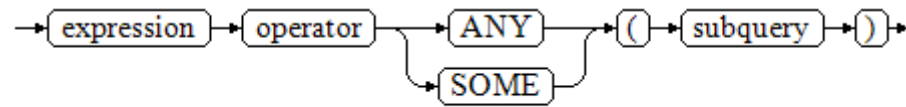
openGauss=# SELECT * FROM in_t1 t1 WHERE t1.a IN (SELECT t2.a FROM in_t2 t2);
a | b
----+----
3 | 4
4 | 5
(2 rows)

openGauss=# 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**.

Example:

```
openGauss=# CREATE TABLE any_t1(a int, b int);
openGauss=# INSERT INTO any_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE any_t2(a int, c int);
openGauss=# INSERT INTO any_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

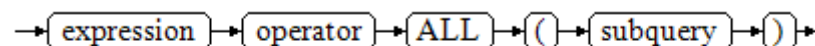
openGauss=# SELECT * FROM any_t1 t1 WHERE t1.a < ANY(SELECT t2.a FROM any_t2 t2 where t2.a =
3 or t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
 3 | 4
(3 rows)

openGauss=# 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.

Example:

```
openGauss=# CREATE TABLE all_t1(a int, b int);
openGauss=# INSERT INTO all_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE all_t2(a int, c int);
openGauss=# INSERT INTO all_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

openGauss=# 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)

openGauss=# 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:

```
openGauss=# 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.

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**.

For example:

```
openGauss=# 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).

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.

```
openGauss=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
result
-----
t
(1 row)
openGauss=# 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.

SOME is a synonym of ANY.

ALL (array)

expression operator ALL (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.

- 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 yields a null array, the result of **ALL** will be null. If the left-hand expression yields null, the result of **ALL** is ordinarily null (though a non-strict comparison operator could possibly yield a different result). Also, 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.

```
openGauss=# 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 fields 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:

```
openGauss=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
result
-----
t
(1 row)

openGauss=# select (4,5,6) > (3,2,1) as result;
result
-----
t
(1 row)

openGauss=# select (4,1,1) > (3,2,1) as result;
result
-----
t
(1 row)

openGauss=# select ('test','data') > ('data','data') as result;
result
-----
t
(1 row)

openGauss=# select (4,1,1) > (3,2,null) as result;
result
-----
t
(1 row)

openGauss=# select (null,1,1) > (3,2,1) as result;
result
-----
(1 row)

openGauss=# select (null,5,6) > (null,5,6) as result;
result
-----
(1 row)

openGauss=# select (4,5,6) > (4,5,6) as result;
result
```

```
-----  
f  
(1 row)  
  
openGauss=# select (2,2,5) >= (2,2,3) as result;  
result  
-----  
t  
(1 row)  
  
openGauss=# 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.

Example:

```
openGauss=# select (1,2,3) = (1,2,3) as result;  
result  
-----  
t  
(1 row)  
  
openGauss=# select (1,2,3) <> (2,2,3) as result;  
result  
-----  
t  
(1 row)  
  
openGauss=# select (2,2,3) <> (2,2,null) as result;  
result  
-----  
  
(1 row)  
  
openGauss=# select (null,5,6) <> (null,5,6) as result;  
result  
-----  
  
(1 row)
```

7.7 Type Conversion

7.7.1 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. For example, the query:

```
openGauss=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
label | value
-----+-----
Origin | (0,0)
(1 row)
```

has two literal constants, of type **text** and **point**. 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**
SQL **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.

7.7.2 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** literals are assumed to be convertible 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.

 CAUTION

After an operator is found, if the type of the input parameter is different from that of the operator, implicit type conversion may occur. After the conversion, unpredictable behavior may occur. If the behavior is incorrect after implicit conversion, you can use explicit type conversion to avoid this problem. For example, after the fixed-length bpchar type is converted to the variable-length text type, spaces at the end of the character string are deleted. If the character string is compared with other character strings, errors may occur.

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 type **bigint**. The scanner assigns an initial type of **bigint** to the argument in this query expression:

```
openGauss=# SELECT 40 ! AS "40 factorial";
          40 factorial
-----
8159152832478977343456112695961158942720000000000
(1 row)
```

So the parser does a type conversion on the operand and the query is equivalent to:

```
openGauss=# 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:

```
openGauss=# 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:

```
openGauss=# 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 type **float8**, which is the preferred type in the numeric category. Therefore, GaussDB will use that entry when faced with an unknown input:

```
openGauss=# SELECT @ '-4.5' AS "abs";
abs
-----
4.5
(1 row)
```

Here 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:

```
openGauss=# 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.)

7.7.3 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** 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** literals are assumed to be convertible 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-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, 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**. So the following query automatically converts the first argument of type **integer** to **numeric**:

```
openGauss=# SELECT round(4, 4);
round
-----
4.0000
(1 row)
```

That query is actually transformed by the parser to:

```
openGauss=# SELECT round(CAST (4 AS numeric), 4);
```

Since numeric constants with decimal points are initially assigned the type **numeric**, the following query will require no type conversion and therefore might be slightly more efficient:

```
openGauss=# 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 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**).

```
openGauss=# SELECT substr('1234', 3);
substr
-----
```

```
34
(1 row)
```

If the string is declared to be of type **varchar**, as might be the case if it comes from a table, then the parser will try to convert it to become **text**:

```
openGauss=# SELECT substr(varchar '1234', 3);
substr
-----
34
(1 row)
```

This is transformed by the parser to effectively become:

```
openGauss=# 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 type **integer**, the parser will try to convert that to **text**:

```
openGauss=# SELECT substr(1234, 3);
substr
-----
34
(1 row)
```

This is transformed by the parser to effectively become:

```
openGauss=# SELECT substr(CAST (1234 AS text), 3);
substr
-----
34
(1 row)
```

7.7.4 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** 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:

```
openGauss=# CREATE SCHEMA tpceds;
openGauss=# CREATE TABLE tpceds.value_storage_t1 (
  VS_COL1 CHARACTER(20)
)DISTRIBUTE BY HASH (VS_COL1);
openGauss=# INSERT INTO tpceds.value_storage_t1 VALUES('abcdef');
openGauss=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpceds.value_storage_t1;
   vs_col1   | octet_length
-----+-----
abcdef      |          20
(1 row)
)
```

```
openGauss=# DROP TABLE tpceds.value_storage_t1;
openGauss=# DROP SCHEMA tpceds;
```

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.

7.7.5 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. The **INTERSECT** and **EXCEPT** construct 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 it is not **unknown**, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text** (the preferred type of the string category). Otherwise, **unknown** inputs are ignored.
- If the inputs are not all of the same type category, a failure will be resulted. (Type **unknown** 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 it is not **unknown**, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text**.
- If inputs are of the string type (including **unknown** which is resolved as type **text**) and digit type, resolve 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-compatible 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. (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 it is not **unknown**, resolve as that 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 blank 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	reltime	abstime
bool	\																								
int1		\	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int8		yes	yes	yes	\	yes	yes	yes		yes	yes	yes	yes	yes											
float4		yes	yes	yes	yes	\	yes	yes		yes	yes	yes	yes	yes											
float8		yes	yes	yes	yes	yes	\	yes		yes	yes	yes	yes	yes											
numeric		yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char		yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar		yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												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
timestampz														yes			yes		\	yes					yes
smalldatetime												yes		yes			yes		yes	yes	\				yes
interval												yes	yes	yes								\	yes		
reltime														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 type **text**.

```
openGauss=# 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 type **numeric**, and the **integer** value **1** can be cast implicitly to **numeric**, so that type is used.

```
openGauss=# 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 type **real** cannot be implicitly cast to **integer**, but **integer** can be implicitly cast to **real**, the union result type is resolved as **real**.

```
openGauss=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
real
-----
```



```
1
2.2
(2 rows)
```

Example 4: In the **TD** type, if input parameters for **COALESCE** are of **int** and **varchar** types, resolve as type **varchar**. In ORA-compatible mode, an error is reported.

```
-- In Oracle-compatible mode, create the ORA-compatible database oracle_1.
openGauss=# CREATE DATABASE oracle_1 dbcompatibility = 'ORA';

-- Switch to the oracle_1 database.
openGauss=# \c oracle_1

-- Create the t1 table.
oracle_1=# CREATE TABLE t1(a int, b varchar(10));
```

```
oracle_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.
openGauss=# CREATE DATABASE td_1 dbcompatibility = 'TD';

-- Switch to the td_1 database.
openGauss=# \c td_1

-- Create the t2 table.
td_1=# CREATE TABLE t2(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for COALESCE.
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 databases in Oracle and TD mode.
openGauss=# DROP DATABASE oracle_1;
openGauss=# DROP DATABASE td_1;
```

Example 5: In ORA-compatible 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-compatible mode, create the ORA-compatible database ora_1.
openGauss=# CREATE DATABASE ora_1 dbcompatibility = 'A';
```

```
-- Switch to the ora_1 database.
openGauss=# \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.
openGauss=# DROP DATABASE ora_1;
DROP DATABASE
```

7.8 System Operation

GaussDB 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** performs 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, 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](#).

7.9 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.

7.10 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 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 7-97](#).

Table 7-97 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 7-98](#).

Table 7-98 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 7-99](#).

Table 7-99 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 7-100](#).

Table 7-100 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 7-101](#).

Table 7-101 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 7-102](#).

Table 7-102 SQL statements for defining an index

Function	SQL Statement
Creating an index	CREATE INDEX
Altering index attributes	ALTER INDEX
Dropping an index	DROP INDEX
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 7-103](#).

Table 7-103 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 7-104](#).

Table 7-104 SQL statements for defining a function

Function	SQL Statement
Creating a function	CREATE FUNCTION
Altering function attributes	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 7-105](#) lists the related SQL statements.

Table 7-105 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 allocates a memory fragment to store the context. Cursors are handles or pointers to the context area. With a cursor, the stored procedure can control alterations in context areas. For details, see [Table 7-106](#).

Table 7-106 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

7.11 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 7-107](#) lists the related SQL statements.

Table 7-107 SQL statements related to sessions

Function	SQL Statement
Altering a session	ALTER SESSION
Killing a session	ALTER SYSTEM KILL SESSION

7.12 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 7-108](#).

Table 7-108 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 7-109](#).

Table 7-109 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](#).

7.13 SQL Syntax

7.13.1 ABORT

Description

Rolls back the current transaction and cancels the changes in the transaction.

This command 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 ] ;
```

Parameters

- WORK | TRANSACTION**

Specifies an optional keyword, which has no effect except increasing readability.

Examples

```
-- Create the customer_demographics_t1 table.
openGauss=# CREATE TABLE customer_demographics_t1
(
  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);

-- Insert data.
openGauss=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200,
'GOOD', 1, 0,0);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Update the column.
openGauss=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

-- Abort the transaction. All updates are rolled back.
openGauss=# ABORT;

-- Query data.
openGauss=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status | cd_purchase_estimate | cd_credit_rating
| cd_dep_count | cd_dep_employed_count | cd_dep_college_count
+-----+-----+-----+-----+-----+-----+-----+
1920801 | M | U | DOCTOR DEGREE | 200 | GOOD | 1
| 0 | 0
(1 row)
```

```
-- Drop the table.
openGauss=# DROP TABLE customer_demographics_t1;
```

Helpful Links

[SET TRANSACTION, COMMIT | END](#), and [ROLLBACK](#)

7.13.2 ALTER AUDIT POLICY

Description

Alters the unified audit policy.

Precautions

- Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.
- The unified audit policy takes effect only after **enable_security_policy** is enabled.

Syntax

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]  
[ access_audit_clause ] };  
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );  
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;  
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;  
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { 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 that complies with the [Identifier Naming Conventions](#).
- **DDL**
Specifies the operations that will be audited within the database: ALTER, ANALYZE, COMMENT, CREATE, DROP, GRANT, REVOKE, SET, SHOW, LOGIN_ACCESS, LOGIN_FAILURE, LOGOUT, and LOGIN.
- **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 operations supported by the specified DDL or DML statements 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: **IP**, **ROLES**, and **APP**.
- **filter_value**
Specifies the detailed information to be filtered.

- **policy_comments**
Records description information of the audit policy.
- **ENABLE|DISABLE**
Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

See [Examples](#) in "CREATE AUDIT POLICY."

Helpful Links

[CREATE AUDIT POLICY](#) and [DROP AUDIT POLICY](#)

7.13.3 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 administrators 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**, thereby resuming cn_5001 in the cluster:

```
ALTER COORDINATOR cn_5001 SET True WITH (cn_5002,cn_5003);
```

7.13.4 ALTER DATABASE

Description

Alters a database, including its name, owner, object isolation, and connection limitation.

Precautions

- Only the database owner or a user granted the ALTER permission can run the **ALTER DATABASE** command. System administrators have this permission by default. The following are permission restrictions depending on attributes to be modified:
 - To change the database name, you must have the CREATEDB permission.
 - To change 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 change the default tablespace of the database, the user must have the CREATE permission for creating tablespaces. 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_NAMESPACES, 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.

Parameters

- **database_name**

Specifies the name of the database whose attributes are to be modified.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **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 that complies with the [Identifier Naming Conventions](#).

- **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** disables the setting.

NOTICE

The current version does not support setting database-level parameters.

Value range: a string.

- **DEFAULT**
- **OFF**
- **RESET**

- **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 "CREATE DATABASE."

Helpful Links

[CREATE DATABASE](#) and [DROP DATABASE](#)

7.13.5 ALTER DEFAULT PRIVILEGES

Description

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

Only the permissions for tables (including views), sequences, functions, and types can be changed.

Syntax

```
ALTER DEFAULT PRIVILEGES
[ FOR { ROLE | USER } target_role [, ...] ]
[ IN SCHEMA schema_name [, ...] ]
abbreviated_grant_or_revoke;
```

- The `abbreviated_grant_or_revoke` clause 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 CEKs.

```
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 ]
```

Parameters

- **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.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

-- Create a common user jack.
openGauss=# CREATE USER jack PASSWORD 'xxxxxxxxxx';

-- Grant the INSERT permission on all the tables in tpcds to the user jack.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

-- Revoke the preceding permissions.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

-- Drop user jack.
openGauss=# DROP USER jack CASCADE;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds;
```

Helpful Links

[GRANT](#) and [REVOKE](#)

7.13.6 ALTER DIRECTORY

Description

Alters 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.  
openGauss=# CREATE OR REPLACE DIRECTORY dir as '/tmp/';  
  
-- Create a user.  
openGauss=# CREATE USER system PASSWORD '*****';  
  
-- Change the owner of the directory.  
openGauss=# ALTER DIRECTORY dir OWNER TO system;  
  
-- Drop a directory.  
openGauss=# DROP DIRECTORY dir;
```

Helpful Links

[CREATE DIRECTORY](#) and [DROP DIRECTORY](#)

7.13.7 ALTER FUNCTION

Description

Alters the attributes of a custom function.

Precautions

Only the function owner or a user granted with the ALTER permission can run the **ALTER FUNCTION** command. System administrators have this permission by default. The following are permission restrictions 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.

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;
```
- **Modify the schema of the user-defined function.**

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype } [, ...] ] )  
    SET SCHEMA new_schema;
```

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**
Specifies the parameter name.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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 a function always returns **NULL** when the value of any of its parameters is **NULL**. If **STRICT** is specified, the function will not be executed when there are null parameters; 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 a system administrator.

- **EXTERNAL**

(Optional) The purpose is to be compatible with SQL. 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. Omitting this parameter is the same as specifying it.

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.

- **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** disables the setting.

Value range: a string.

- DEFAULT
- OFF
- RESET
- Specified 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 that complies with the [Identifier Naming Conventions](#).

- **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.

Value range: an existing user role.

- **new_schema**

Specifies the new schema of a function.

Value range: an existing schema.

Examples

See [Examples](#) in "CREATE FUNCTION."

Helpful Links

[CREATE FUNCTION](#) and [DROP FUNCTION](#)

7.13.8 ALTER GLOBAL CONFIGURATION

Description

Adds or changes 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.
openGauss=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = true);

-- Query.
openGauss=# SELECT * FROM gs_global_config;
      name      | value
-----+-----
 buckets_len    | 16384
 undorstoragetype | page
 redis_is_ok     | true
(3 rows)

-- Alter.
openGauss=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = false);

-- Query.
openGauss=# SELECT * FROM gs_global_config;
      name      | value
-----+-----
 buckets_len    | 16384
 undorstoragetype | page
 redis_is_ok     | false
(3 rows)

-- Drop.
openGauss=# DROP GLOBAL CONFIGURATION redis_is_ok;

-- Query.
openGauss=# SELECT * FROM gs_global_config;
      name      | value
-----+-----
 buckets_len    | 16384
 undorstoragetype | page
(2 rows)
```

Helpful Links

[DROP GLOBAL CONFIGURATION](#)

7.13.9 ALTER GROUP

Description

Alters the attributes of a user group.

Precautions

ALTER GROUP is an alias for ALTER ROLE, and it is not a standard SQL command 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;`

Parameters

See [Parameters](#) in "ALTER ROLE."

Examples

```
-- Create a user group.
openGauss=# CREATE GROUP super_users WITH PASSWORD '*****';

-- Create a user.
openGauss=# CREATE ROLE lche WITH PASSWORD '*****';

-- Create a user.
openGauss=# CREATE ROLE jim WITH PASSWORD '*****';

-- Add users to a group.
openGauss=# ALTER GROUP super_users ADD USER lche, jim;

-- Remove users from a group.
openGauss=# ALTER GROUP super_users DROP USER jim;

-- Change the name of the group.
openGauss=# ALTER GROUP super_users RENAME TO normal_users;

-- Drop users.
openGauss=# DROP ROLE lche, jim;

-- Drop the user group.
openGauss=# DROP GROUP normal_users;
```

Helpful Links

[CREATE GROUP](#), [DROP GROUP](#), and [ALTER ROLE](#)

7.13.10 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

This option 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 or a user who has the INDEX permission on the table where the index resides can run the **ALTER INDEX** command. System administrators have this permission by default.

Syntax

- Rename a table index.

```
ALTER INDEX [ IF EXISTS ] index_name  
  RENAME TO new_name;
```
- Alter 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 that complies with the [Identifier Naming Conventions](#).
- **tablespace_name**
Specifies the tablespace name.
Value range: an existing tablespace name.
- **storage_parameter**
Specifies the name of an index-method-specific parameter.
- **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 an index partition.
- **index_partition_name**
Specifies the name of an index partition.
- **new_tablespace**
Specifies a new tablespace.

Examples

See [Examples](#) in "CREATE INDEX."

Helpful Links

[CREATE INDEX](#), [DROP INDEX](#), and [REINDEX](#)

7.13.11 ALTER LANGUAGE

Description

Modifies the definition of a procedural language.

Precautions

The current version does not support this syntax.

7.13.12 ALTER MASKING POLICY

Description

Alters 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 **enable_security_policy** is enabled.

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 policy takes effect.
`ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...]*)[, ...]*);`
- Remove the filters of the masking policy.
`ALTER MASKING POLICY policy_name DROP FILTER;`
- Enable or disable the masking policy.
`ALTER MASKING POLICY policy_name [ENABLE | DISABLE];`

Parameters

- **policy_name**
Specifies the name of a masking policy, which must be unique.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **policy_comments**
Adds or modifies description of the masking policy.
- **masking_function**
Specifies eight preset masking methods or user-defined functions. Schemas are supported.
The maskall function is not preset. It is hard-coded and cannot be displayed by running `\df`.
The masking methods during presetting are as follows:
`maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking | shufflemasking | alldigitsmasking | regexpmasking`
- **label_name**
Specifies the resource label name.
- **FILTER_TYPE**
Specifies the types of information to be filtered by the masking policy, including **IP**, **ROLES**, and **APP**.
- **filter_value**
Specifies 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.
openGauss=# CREATE USER dev_mask PASSWORD 'xxxxxxxxxx';
openGauss=# CREATE USER bob_mask PASSWORD 'xxxxxxxxxx';

-- Create table tb_for_masking.
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

-- Create a resource label for sensitive column col1.
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a resource label for sensitive column col2.
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a masking policy for the operation of accessing sensitive column col1.
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Add description for masking policy maskpol1.
openGauss=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';

-- Modify masking policy maskpol1 to add a masking method.
openGauss=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);

-- Modify masking policy maskpol1 to remove a masking method.
openGauss=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);

-- Modify masking policy maskpol1 to modify a masking method.
openGauss=# 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 gsql, and the IP addresses are 10.20.30.40 and 127.0.0.0/24.
openGauss=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask), APP(gsql), IP('10.20.30.40', '127.0.0.0/24'));

-- Modify masking policy maskpol1 so that it takes effect for all user scenarios.
openGauss=# ALTER MASKING POLICY maskpol1 DROP FILTER;

-- Disable masking policy maskpol1.
openGauss=# ALTER MASKING POLICY maskpol1 DISABLE;

-- Drop a masking policy.
openGauss=# DROP MASKING POLICY maskpol1;

-- Drop resource labels.
openGauss=# DROP RESOURCE LABEL mask_lb1, mask_lb2;

-- Drop the tb_for_masking table.
openGauss=# DROP TABLE tb_for_masking;

-- Drop the dev_mask and bob_mask users.
openGauss=# DROP USER dev_mask, bob_mask;
```

Helpful Links

[CREATE MASKING POLICY](#) and [DROP MASKING POLICY](#)

7.13.13 ALTER MATERIALIZED VIEW

Description

Alters 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 MATERIALIZED VIEW permission.
- The materialized view structure cannot be modified.

Syntax

- Alter the owner of the 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;
```

Parameters

- **mv_name**
Specifies the name of an existing materialized view, which can be schema-qualified.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **column_name**
Specifies the name of a new or existing column.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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.  
openGauss=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);  
  
-- Create a complete-refresh materialized view.  
openGauss=# CREATE MATERIALIZED VIEW foo AS SELECT * FROM my_table;  
  
-- Rename the materialized view foo to bar.  
openGauss=# ALTER MATERIALIZED VIEW foo RENAME TO bar;  
  
-- Drop a complete-refresh materialized view.  
openGauss=# DROP MATERIALIZED VIEW bar;  
  
-- Drop the my_table table.  
openGauss=# DROP TABLE my_table;
```

Helpful Links

[CREATE INCREMENTAL MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#),
[DROP MATERIALIZED VIEW](#), [REFRESH INCREMENTAL MATERIALIZED VIEW](#),
and [REFRESH MATERIALIZED VIEW](#)

7.13.14 ALTER NODE

Function

ALTER NODE modifies the definition of an existing node.

Precautions

ALTER NODE is an interface of the cluster management tool and is used to manage clusters. Only administrators have the permission to use this interface. You are not advised to use this interface, 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.

Parameter Description

See [Parameters](#) in **CREATE NODE**.

Helpful Links

[CREATE NODE](#) and [DROP NODE](#)

7.13.15 ALTER NODE GROUP

Description

Alters the 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 that complies with the [Identifier Naming Conventions](#).
- **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_name** in the **pggroup** columns of the pgxc_class system catalogs 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.
- **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.

- **RESIZE TO dest_group_name**

Specifies a resize flag for the cluster. Set **groupname** to the source node group before data redistribution and **is_installation** of the node group to **FALSE**. Set **desst_group_name** to the destination node group and **is_installation** of the node group to **TRUE**.

Helpful Links

[CREATE NODE GROUP](#) and [DROP NODE GROUP](#)

7.13.16 ALTER RESOURCE LABEL

Description

Alters 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 that complies with the [Identifier Naming Conventions](#).
- **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.
openGauss=# CREATE TABLE table_for_label(col1 int, col2 text);

-- Create resource label table_label.
openGauss=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

-- Attach resource label table_label to col2.
openGauss=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2)

-- Remove table_label from an item.
```



```
openGauss=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

-- Drop the resource label table_label.
openGauss=# DROP RESOURCE LABEL table_label;

-- Drop the base table table_for_label.
openGauss=# DROP TABLE table_for_label;
```

Helpful Links

[CREATE RESOURCE LABEL](#) and [DROP RESOURCE LABEL](#)

7.13.17 ALTER ROLE

Description

Alters 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 | NOVCAADMIN}
| {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 the setting of 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. A system administrator cannot reset passwords of other system administrators.
- **EXPIRED**
Invalidates 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 an administrator resets the password.

For details about other parameters, see [Parameters](#) in "CREATE ROLE."

Examples

See [Examples](#) in "CREATE ROLE."

Helpful Links

[CREATE ROLE](#), [DROP ROLE](#), and [SET ROLE](#)

7.13.18 ALTER ROW LEVEL SECURITY POLICY

Description

Alters 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

```
ALTER [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name RENAME TO  
new_policy_name;  
  
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 an expression defined for a row-level security policy. The return value is of the Boolean type.

Examples

```
-- Create data table all_data.
openGauss=# 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.
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);
openGauss=# \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"
    USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row

-- Change the name of the all_data_rls policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

-- Create users alice and bob.
openGauss=# CREATE ROLE alice WITH PASSWORD '*****';
openGauss=# CREATE ROLE bob WITH PASSWORD '*****';

-- Change the users affected by the row-level security policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
openGauss=# \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_new_rls"
    TO alice,bob
    USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, enable_rowsecurity=true

-- Modify the expression defined for the row-level security policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role
= current_user);
openGauss=# \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_new_rls"
  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, enable_rowsecurity=true

-- Drop users alice and bob.
openGauss=# DROP ROLE alice, bob;

-- Drop the policy.
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data;

-- Drop the all_data table.
openGauss=# DROP TABLE all_data;
```

Helpful Links

[CREATE ROW LEVEL SECURITY POLICY](#) and [DROP ROW LEVEL SECURITY POLICY](#)

7.13.19 ALTER SCHEMA

Description

Alters 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 a system administrator and a member of the new owner role.
- Only the initial user is allowed to change the owner of the pg_catalog system 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**.

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**

Renames a schema. If a non-SYSADMIN user wants to change the schema name, the user must have the CREATE permission on the database.

new_name: new name of the schema.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **OWNER TO new_owner**

Changes the owner of a schema. To do this as a non-system 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.
openGauss=# CREATE SCHEMA ds;

-- Rename the current schema ds to ds_new.
openGauss=# ALTER SCHEMA ds RENAME TO ds_new;

-- Create user jack.
openGauss=# CREATE USER jack PASSWORD '*****';

-- Change the owner of ds_new to jack.
openGauss=# ALTER SCHEMA ds_new OWNER TO jack;

-- Drop user jack and schema ds_new.
openGauss=# DROP SCHEMA ds_new;
openGauss=# DROP USER jack;
```

Helpful Links

[CREATE SCHEMA](#) and [DROP SCHEMA](#)

7.13.20 ALTER SEQUENCE

Description

Alters the parameters of an existing sequence.

Precautions

- Only the sequence owner or a user granted the ALTER permission 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 a 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 change other parameters, drop the sequence and create it again. Then, use the Setval function to restore parameter values.
- ALTER SEQUENCE MAXVALUE cannot be used in transactions, functions, or 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 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 nonexistent sequence.
- **OWNED BY**
Joins a sequence with a specified column included in a table. In this way, the sequence will be dropped when you drop its joined column or the table where the column belongs to.

If the sequence has been joined with another table before you use this option, the new join will overwrite the old one.

The joined table and sequence must be owned by the same user and in the same schema.

If **OWNED BY NONE** is used, all existing joins will be dropped.
- **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, starting from 101.
openGauss=# CREATE SEQUENCE serial START 101;

-- Create a table and specify default values for the sequence.
openGauss=# CREATE TABLE T1(C1 bigint default nextval('serial'));

-- Change the owning column of serial to T1.C1.
openGauss=# ALTER SEQUENCE serial OWNED BY T1.C1;

-- Drop the sequence.
openGauss=# DROP SEQUENCE serial cascade;
openGauss=# DROP TABLE T1;
```

Helpful Links

[CREATE SEQUENCE](#) and [DROP SEQUENCE](#)

7.13.21 ALTER SERVER

Description

Adds, modifies, 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 a 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 latest version of the server.
- OPTIONS**
Changes options of the server. **ADD**, **SET**, and **DROP** are operations to be performed. If the operation is not specified explicitly, **ADD** operations will be performed. 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**: Data is encrypted.
- **off**: 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, the AK value is encrypted and saved to the metadata table of the database. This parameter is available only when **type** is set to **OBS**.
- **secret_access_key**
Specifies the SK value (obtained by users from the OBS console) used for the OBS access protocol. When you create a foreign table, the SK value is encrypted and saved to the metadata table of the database. 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 foreign data wrapper of the foreign server.
- **new_name**
Specifies the new name of the server.

Examples

```
-- Create my_server.  
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;  
  
-- Change the name of a foreign server.  
openGauss=# ALTER SERVER my_server RENAME TO my_server_1;  
  
-- Drop my_server_1.  
openGauss=# DROP SERVER my_server_1;
```

Helpful Links

[CREATE SERVER](#) and [DROP SERVER](#)

7.13.22 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.

Syntax

- Set transaction parameters of a session.

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION  
{ ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED } | { 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

- **SESSION**
Specifies that the specified parameters take effect for the current session. If neither **SESSION** nor **LOCAL** appears, **SESSION** is the default value.
If this command is executed in a transaction, the effect of the command disappears after the transaction is rolled back. Once the surrounding transaction is committed, the effect will persist until the end of the session, unless the parameters are reset by another SET statement.
- **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**.
- **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.
- **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 GUC 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) and **DOCUMENT**.
- **config_parameter**
Specifies the name of a configurable GUC parameter. You can use **SHOW ALL** to view the available GUC parameters.

Examples

```
-- Create the ds schema.
openGauss=# CREATE SCHEMA ds;

-- Set the search path of the schema.
openGauss=# SET SEARCH_PATH TO ds, public;

-- Set the time/date type to the traditional Postgres format (date before month).
openGauss=# SET DATESTYLE TO postgres, dmy;

-- Set the character code of the current session to UTF8.
openGauss=# ALTER SESSION SET NAMES 'UTF8';

-- Set the time zone to Berkeley of California.
openGauss=# SET TIME_ZONE 'PST8PDT';

-- Set the time zone to Italy.
openGauss=# SET TIME_ZONE 'Europe/Rome';

-- Set the current schema.
openGauss=# ALTER SESSION SET CURRENT_SCHEMA TO tpceds;

-- Set XML OPTION to DOCUMENT.
openGauss=# ALTER SESSION SET XML OPTION DOCUMENT;

-- Create the role joe, and set the session role to joe.
openGauss=# CREATE ROLE joe WITH PASSWORD '*****!';
openGauss=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '*****!';

-- Switch to the default user.
openGauss=> ALTER SESSION SET SESSION AUTHORIZATION default;

-- Drop the ds schema.
openGauss=# DROP SCHEMA ds;

-- Drop the role joe.
openGauss=# DROP ROLE joe;

-- Start a transaction and set the transaction level.
openGauss=# START TRANSACTION;
openGauss=# ALTER SESSION SET TRANSACTION READ ONLY;
openGauss=# END;
```

Helpful Links

[SET](#)

7.13.23 ALTER SYNONYM

Description

Alters 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 that complies with the [Identifier Naming Conventions](#).
- **new_owner**
Specifies the new owner of the SYNONYM object.
Value range: a string. It must be a valid username.

Examples

```
-- Create synonym t1.  
openGauss=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;  
  
-- Create user u1.  
openGauss=# CREATE USER u1 PASSWORD '*****';  
  
-- Change the owner of synonym t1 to u1.  
openGauss=# ALTER SYNONYM t1 OWNER TO u1;  
  
-- Drop synonym t1.  
openGauss=# DROP SYNONYM t1;  
  
-- Drop user u1.  
openGauss=# DROP USER u1;
```

Helpful Links

[CREATE SYNONYM](#) and [DROP SYNONYM](#)

7.13.24 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).
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.
openGauss=# 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.
openGauss=# ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

7.13.25 ALTER TABLE

Description

Alters 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

- The owner of a table, users granted the ALTER permission on the table, or users granted 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 a system administrator and a member of the new owner role.
- The tablespace of a partitioned table cannot be modified, but the tablespace of a 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 **PrepareStatement**.

- 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 a **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**, **TIMESTAMPZ**, **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'.

Syntax

- Modify the definition of a table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
action [, ... ];
```

The table operation ACTION can be one of the following clauses:

```
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  
| 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  
| DISABLE/ENABLE [ REPLICA | ALWAYS ] RULE  
| DISABLE ROW LEVEL SECURITY  
| ENABLE ROW LEVEL SECURITY  
| FORCE ROW LEVEL SECURITY  
| NO FORCE ROW LEVEL SECURITY  
  
| INHERIT parent_table  
| NO INHERIT parent_table  
| DISTRIBUTE BY { REPLICATION | { [ HASH ] ( column_name ) } }  
| OF type_name  
| NOT OF  
| REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```

 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.
- **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]**
Drops 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.
- **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**.
- **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 scale-out 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 fired only when the session is in replica mode.
- **| ENABLE ALWAYS TRIGGER trigger_name**
Determines that all triggers are fired regardless of the current replication mode.
- **| { DISABLE | ENABLE } [REPLICA | ALWAYS] RULE**
Enables or disables a rule for tables. Disabled rules are still visible in the system, but are not applied during query rewriting. The ON SELECT rule cannot be disabled because it is related to the view implementation. Rules configured as **ENABLE REPLICA** are enabled only when the session is in replica mode, while those configured as **ENABLE ALWAYS** can be enabled regardless of the replica mode. Rule triggering is also affected by configuration variable *session_replication_role*, which is similar to the preceding trigger setting.
- **| { 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.
- **INHERIT parent_table**
Adds the target data table to a specified parent data table as a new child data table. After that, the query for the parent data table will contain the data in the target data table. Before being added as a child data table, the target data table must contain all the columns in the parent data table. These columns must have matching data categories, and if they have NOT NULL constraints in the parent data table, they must also have NOT NULL constraints in the child data table. For all CHECK constraints in the parent data table, there must be corresponding constraints in the child data table, unless the parent data table is marked as non-inheritable.
- **NO INHERIT parent_table**
Generates the target data table from the child data table of a specified parent data table. Queries for the parent data table will no longer contain records generated from the target data table.
- **DISTRIBUTE BY { REPLICATION | { [HASH] (column_name) } }**
Specifies how the table is distributed or replicated between nodes.
- **OF type_name**
Joins a table to a composite type, which is similar to table creation by using the **CREATE TABLE OF** option. The name and type of a table column must exactly match those defined in the composite type, but the OID system column can be

different. The table cannot be inherited from any other table. These restrictions ensure that the **CREATE TABLE OF** option allows the same table definition.

- **NOT OF**

Removes the association between a table and a type.

- **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.

- There are several clauses of **column_clause**:

```
ADD [ COLUMN ] column_name data_type [ 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 [COLLATE collation] [column_constraint [...]]**

Adds a column to a table. 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).

- **ADD ({ column_name data_type } [, ...])**

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 on the table will store a **NULL** value for the column. Therefore, dropping a column takes a short period of time but does not immediately release the tablespace on the disks, because the space occupied by the dropped 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]**

Alters 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 drops 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**

Alters 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 non-**NULL** values.

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

Specifies the per-column statistics-collection 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.

- **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. This clause specifies whether this column is held inline or in a secondary TOAST table, and whether the data should be compressed. Only row-store tables can be set. SET STORAGE itself does not change anything in the table. It sets the strategy to be pursued during future table updates.

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
  [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **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 |
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

index_parameters is as follows:

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- 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 [ 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 name of an index.
- **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** parameters are used only by online scaling 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.
It can be used only for data redistribution.
- **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.

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 this function 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 system catalog. 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.
- **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.
- **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 value, 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 that 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 index_parameters**

UNIQUE (column_name [, ...]) index_parameters

Specifies that a group of one or more columns of a table can contain only unique values.

- **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.
- **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 or constraint if there are any dependent objects. This is the default behavior.
- **schema_name**
Specifies the schema name of a table.

Examples of Altering a Table

- Rename a table.

```
openGauss=# CREATE TABLE aa(c1 int, c2 int);
openGauss=# ALTER TABLE IF EXISTS aa RENAME TO test_alt1;
```
- Alter the schema of a table.
-- Create **test_schema**.

```
openGauss=# CREATE SCHEMA test_schema;
```


-- Change the schema of the **test_alt1** table to **test_schema**.

```
openGauss=# ALTER TABLE test_alt1 SET SCHEMA test_schema;
```


-- Query table information.

```
openGauss=# 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 **test_user**.

```
openGauss=# CREATE USER test_user PASSWORD 'XXXXXXXXXX';
```


-- Change the owner of the **test_alt1** table to **test_user**.

```
openGauss=# ALTER TABLE IF EXISTS test_schema.test_alt1 OWNER TO test_user;
```


-- Query.

```
openGauss=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename = 'test_alt1';
tablename | schemaname | tableowner
-----+-----+-----
test_alt1 | test_schema | test_user
(1 row)
```
- Alter the tablespace of a table.
-- Create the **tbs_data1** tablespace.

```
openGauss=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';
```


-- Change the tablespace of the **test_alt1** table to **tbs_data1**.

```
openGauss=# ALTER TABLE test_schema.test_alt1 SET TABLESPACE tbs_data1;
```


-- Query.

```
openGauss=# SELECT tablename, tablespace FROM pg_tables WHERE tablename = 'test_alt1';
tablename | tablespace
```



```
-----+-----  
test_alt1 | tbs_data1  
(1 row)  
  
-- Drop.  
openGauss=# DROP TABLE test_schema.test_alt1;  
openGauss=# DROP TABLESPACE tbs_data1;  
openGauss=# DROP SCHEMA test_schema;  
openGauss=# DROP USER test_user;
```

Examples of Changing a Column

- Change column names.

```
-- Create a table.  
openGauss=# CREATE TABLE test_alt2(c1 INT,c2 INT);  
-- Change column names.  
openGauss=# ALTER TABLE test_alt2 RENAME c1 TO id;  
openGauss=# ALTER TABLE test_alt2 RENAME COLUMN c2 to areaid;  
-- Query.  
openGauss=# \d test_alt1  
Table "public.test_alt1"  
Column | Type | Modifiers  
-----+-----  
id | integer |  
areaid | integer |
```

- Add a column.

```
-- Add a column to the test_alt1 table.  
openGauss=# ALTER TABLE IF EXISTS test_alt2 ADD COLUMN name VARCHAR(20);  
-- Query.  
openGauss=# \d test_alt2  
Table "public.test_alt1"  
Column | Type | Modifiers  
-----+-----  
id | integer |  
areacode | integer |  
name | character varying(20) |
```

- Alter the data type of a column.

```
-- Change the type of the name column in the test_alt1 table.  
openGauss=# ALTER TABLE test_alt1 MODIFY name VARCHAR(50);  
-- Query.  
openGauss=# \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.  
openGauss=# ALTER TABLE test_alt2 ALTER COLUMN name TYPE VARCHAR(25);  
-- Query.  
openGauss=# \d test_alt2  
Table "public.test_alt2"  
Column | Type | Modifiers  
-----+-----  
id | integer |  
areaid | integer |  
name | character varying(25) |
```

- Drop a column.

```
-- Drop the areaid column from test_alt1.  
openGauss=# ALTER TABLE test_alt2 DROP COLUMN areaid;  
-- Query.  
openGauss=# \d test_alt2  
Table "public.test_alt2"  
Column | Type | Modifiers  
-----+-----  
id | integer |  
name | character varying(25) |
```

- Modify the column-store mode.

```
-- View table details.
openGauss=# \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, storage_type=USTORE

-- Change the storage mode of the name column in the test_alt2 table.
openGauss=# ALTER TABLE test_alt2 ALTER COLUMN name SET STORAGE PLAIN;

-- Query.
openGauss=# \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, storage_type=USTORE

-- Drop.
openGauss=# DROP TABLE test_alt2;
```

Examples of Changing a Constraint

- Add a NOT NULL constraint to a column.

```
-- Create a table.
openGauss=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));
-- Add a NOT NULL constraint to pid.
openGauss=# ALTER TABLE test_alt3 MODIFY pid NOT NULL;
-- Query.
openGauss=# \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.

```
openGauss=# ALTER TABLE test_alt3 MODIFY pid NULL;
-- Query.
openGauss=# \d test_alt3
               Table "public.test_alt3"
  Column |      Type      | Modifiers 
-----+-----+-----
 pid     | integer         | 
 areaid  | character(5)    | 
 name    | character varying(20) |
```

- Alter the default value of a column.

```
-- Alter the default value of id in the test_alt1 table.
openGauss=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';
-- Query.
openGauss=# \d test_alt3
               Table "public.test_alt3"
  Column |      Type      | Modifiers 
-----+-----+-----
 pid     | integer         | 
 areaid  | character(5)    | default '00000':bpchar
 name    | character varying(20) | 
-- Drop the default value of id.
openGauss=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;
-- Query.
openGauss=# \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.
openGauss=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);
-- Query.
openGauss=# \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 a constraint.

```
-- Create a table.
openGauss=# CREATE TABLE test_alt4(c1 INT, c2 INT);
-- Create an index.
openGauss=# CREATE UNIQUE INDEX pk_test4_c1 ON test_alt4(c1);
-- Associate the created index when adding a constraint.
openGauss=# ALTER TABLE test_alt4 ADD CONSTRAINT pk_test4_c1 PRIMARY KEY USING
INDEX pk_test4_c1;
-- Query.
openGauss=# \d test_alt4
      Table "public.test_alt4"
  Column | Type | Modifiers
  -----+-----+-----
 c1      | int  | not null
 c2      | int  |
Indexes:
 "pk_test4_c1" PRIMARY KEY, btree (c1) TABLESPACE pg_default
-- Drop.
openGauss=# DROP TABLE test_alt4;
```

- Drop a table-level constraint.

```
-- Drop a constraint.
openGauss=# ALTER TABLE test_alt3 DROP CONSTRAINT IF EXISTS pk_test3_pid;
-- Query.
openGauss=# \d test_alt3
      Table "public.test_alt3"
  Column |      Type      | Modifiers
  -----+-----+-----
 pid    | integer        | not null
 areaid | integer        |
 name   | character varying(20) |
-- Drop.
openGauss=# DROP TABLE test_alt3;
```

Helpful Links

[CREATE TABLE](#) and [DROP TABLE](#)

7.13.26 ALTER TABLE PARTITION

Description

Alters table partitions, including adding, deleting, splitting, merging, clearing, exchanging, and renaming partitions, moving partition tablespaces, and modifying partition attributes.

Precautions

- Only the owner of a partitioned table or users granted the ALTER permission on the partitioned table can run the **ALTER TABLE PARTITION** command. System administrators have the permission to run the command 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 partition key of the added partition must be the same type as that of the partitioned table. The key value of the added partition must exceed the upper limit of the last partition range.
- If the number of partitions in the target partitioned table reaches the maximum (**1048575**), no more partitions can 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 number of columns specified when you define a partition, and they must be consistent.
- The **Value** partitioned table does not support the ALTER PARTITION operation.
- 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

- 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 runs **DROP PARTITION** and then **ADD PARTITION**, and finally runs 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, and storage parameter.
- The distribution key information of the ordinary table and partition should be consistent.
- The number and information of indexes of the ordinary table and partition should be consistent.
- 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 masking policy.

When the exchange is done, the data and tablespace of the ordinary table and partition are exchanged. The statistics about ordinary tables and partitions become unreliable, and they should be analyzed again. If the DROP COLUMN operation is performed on an ordinary or partitioned table, the dropped column still exists physically. Therefore, you need to ensure that the dropped 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.

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name  
[ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

- 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 ]
```

- The syntax of split_point_clause 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 syntax of **no_split_point_clause** 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 maximum of 4 partition keys, while **partition_start_end_item** supports only one partition key. For details about the supported data types, see • [PARTITION BY RANGE\(partition_key\)](#).
- **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement.

- The syntax of **partition_less_than_item** 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 tablespace_name]
```

- The **add_clause** syntax is used to add one or more partitions to a specified partitioned table.

```
ADD PARTITION ( partition_col1_name = partition_col1_value [, partition_col2_name =
partition_col2_value ] [, ...] )
  [ LOCATION 'location1' ]
  [ PARTITION (partition_colA_name = partition_colA_value [, partition_colB_name =
partition_colB_value ] [, ...] ) ]
  [ LOCATION 'location2' ]
ADD {partition_less_than_item | partition_start_end_item}
```

- 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 ]
```

- Alter the name of a partition.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  RENAME PARTITION { partition_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

Parameters

- **table_name**

Specifies the name of a partitioned table.

Value range: an existing partitioned 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 renamed.
- **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 MOVEMENT**
Specifies whether to enable 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**: 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 ordinary 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 into the correct partition. If there is no correct partition where the data can be inserted, 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 that complies with the [Identifier Naming Conventions](#).
- **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 "CREATE TABLE PARTITION."

Helpful Links

[CREATE TABLE PARTITION](#) and [DROP TABLE](#)

7.13.27 ALTER TABLESPACE

Description

Alters 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 a 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 that complies with the [Identifier Naming Conventions](#).
- **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 "CREATE TABLESPACE."

Helpful Links

[CREATE TABLESPACE](#) and [DROP TABLESPACE](#)

7.13.28 ALTER TRIGGER

Function

ALTER TRIGGER renames a trigger.

 NOTE

Currently, only the name can be modified.

Precautions

Only the owner of a table where the trigger is created and a system administrator can run the **ALTER TRIGGER** statement.

Syntax

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

Parameter Description

- **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 that complies with the identifier 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](#)

7.13.29 ALTER TYPE

Description

Alters the definition of a type.

Precautions

Only the type owner or a user granted with the ALTER permission 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 a new attribute to a composite type.
`ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]`
 - Drop an attribute from a composite type.
`DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]`
 - Alter the type of an attribute in a composite type.
`ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]`
- Add a new attribute to a composite type.
`ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];`
- Drop an attribute from a composite type.
`ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ];`
- Alter the type of an attribute in a composite type.
`ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];`
- Alter the owner of a type.
`ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };`
- Alter the name of a type.
`ALTER TYPE name RENAME TO new_name;`
- Alter the name of an attribute in a composite type.
`ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE | RESTRICT ];`

- 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 ];`
- Alter 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 change 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 user can only rebuild and drop the type. However, system administrators can change ownership 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.

Examples

See [Examples](#) in "CREATE TYPE."

Helpful Links

[CREATE TYPE](#) and [DROP TYPE](#)

7.13.30 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 | NOVCAADMIN }
| { 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 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.

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

See [Examples](#) in "CREATE USER."

Helpful Links

[CREATE ROLE](#), [CREATE USER](#), and [DROP USER](#)

7.13.31 ALTER VIEW

Description

ALTER VIEW alters the auxiliary attributes of a view. If you want to change 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. System administrators have this permission by default. The following are permission restrictions depending on attributes to be modified:

- To modify the schema of a view, you must be the owner of the view or a 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 a 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 [ , ... ] );
```

Parameters

- **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 that complies with the [Identifier Naming Conventions](#).
- **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 that complies with the [Identifier Naming Conventions](#).
- **SET/DROP DEFAULT**
Sets or drops 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.
Currently, **view_option_name** supports only the **security_barrier** parameter. Use this parameter when the view provides row-level security.
Value range: Boolean type. It can be **TRUE** or **FALSE**.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer table.
openGauss=# CREATE TABLE tpcds.customer
(
  c_customer_sk      INTEGER      NOT NULL,
  c_customer_id      CHARACTER(16) NOT NULL
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAAABAAAAAA'),(100,
'AAAAAAACAAAAAA'),(150, 'AAAAAADAAAAAA');

-- Create a view consisting of rows with c_customer_sk less than 150.
openGauss=# CREATE VIEW tpcds.customer_details_view_v1 AS
  SELECT * FROM tpcds.customer
  WHERE c_customer_sk < 150;

-- Rename a view.
openGauss=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;

-- Change the schema of a view.
openGauss=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;
```



```
-- Drop a view.  
openGauss=# DROP VIEW public.customer_details_view_v2;  
  
-- Drop the tpcds.customer table.  
openGauss=# DROP TABLE tpcds.customer;  
  
-- Drop the schema.  
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[CREATE VIEW](#) and [DROP VIEW](#)

7.13.32 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 determine which one is 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** is used to check whether data files of ordinary tables (row-store 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 does not continue 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. By default, system administrators have this permission. 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.

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.

- 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 and cudes 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 from 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**

For 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.

- **foreign table name**

Specifies the name of the specific table to be analyzed (possibly schema-qualified).

Value range: an existing table name.

- **index name**

Specifies the name of the specific index table to be analyzed (possibly schema-qualified).

Value range: an existing table name.

- **FAST|COMPLETE**

For a row-store table, the **FAST** mode verifies the CRC and page header of the row-store table. If the verification fails, an alarm is generated. In **COMPLETE** mode, the pointer and tuple of the row-store table are parsed and verified.

- **CASCADE**

In **CASCADE** mode, all indexes of the current table are verified.

Examples

```
-- Create a table.
```

```
openGauss=# 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.

```
openGauss=# 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;
```

-- Use ANALYZE to update statistics.

```
openGauss=# ANALYZE customer_info;
```

-- Use ANALYZE VERBOSE to update statistics and display table information.

```
openGauss=# ANALYZE VERBOSE customer_info;  
INFO: analyzing "cstore.pg_delta_3394584009"(cn_5002 pid=53078)  
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 the CN.

-- Drop the table.

```
openGauss=# DROP TABLE customer_info;  
openGauss=# DROP TABLE customer_par;
```

7.13.33 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 | READ UNCOMMITTED | 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.  
openGauss=# BEGIN  
dbe_output.print_line('Hello');  
END;  
/
```

Helpful Links

[START TRANSACTION](#)

7.13.34 CALL

Description

CALL 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.
- **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](#). For an overloaded package function, the parameter list can have no output parameter, but the function may not be found. If an output parameter is contained, it must be a constant.

Examples

```
-- Create a function named func_add_sql, calculate the sum of two integers, and return the result.
openGauss=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Pass by parameter value.
openGauss=# CALL func_add_sql(1, 3);

-- Pass by naming tag method.
openGauss=# CALL func_add_sql(num1 => 1,num2 => 3);
openGauss=# CALL func_add_sql(num2 := 2, num1 := 3);

-- Drop the function.
openGauss=# DROP FUNCTION func_add_sql;

-- Create a function with output parameters.
openGauss=# 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.
openGauss=# CALL func_increment_sql(1,2,1);

-- Specify a variable as an output parameter.
openGauss=# DECLARE
res int;
```

```
BEGIN
func_increment_sql(1, 2, res);
dbe_output.print_line(res);
END;
/

-- Create an overloaded function.
openGauss=# CREATE OR REPLACE PROCEDURE package_func_overload(col int, col2 out int) PACKAGE
AS
DECLARE
    col_type text;
BEGIN
    col := 122;
    dbe_output.print_line('two out parameters ' || col2);
END;
/

openGauss=# CREATE OR REPLACE PROCEDURE package_func_overload(col int, col2 out varchar) PACKAGE
AS
DECLARE
    col_type text;
BEGIN
    col2 := '122';
    dbe_output.print_line('two varchar parameters ' || col2);
END;
/

-- Call the function.
openGauss=# CALL package_func_overload(1, 'test');
openGauss=# CALL package_func_overload(1, 1);

-- Drop functions.
openGauss=# DROP FUNCTION package_func_overload;
openGauss=# DROP FUNCTION func_increment_sql;
```

Helpful Links

[CREATE FUNCTION](#) and [CREATE PROCEDURE](#)

7.13.35 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.

It sets transaction log checkpoints. 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** performs 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.  
openGauss=# CHECKPOINT;
```

7.13.36 CLEAN CONNECTION

Description

Clears 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 dropped.
- **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**

Deletes the idle or invalid connections between the current CN 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.
openGauss=# CREATE DATABASE test_clean_connection;

-- Create user jack.
openGauss=# CREATE USER jack PASSWORD '*****';

-- Clean the idle and invalid connections between the current CN and dn1 and dn2 in the template1
database.
openGauss=# CLEAN CONNECTION TO NODE (dn_6001_6002,dn_6003_6004) FOR DATABASE template1;

-- Clean the idle and invalid connections between the current CN and dn1 that are related to user jack.
openGauss=# CLEAN CONNECTION TO NODE (dn_6001_6002) TO USER jack;

-- Clean the connections between the current CN and all nodes related to the test_clean_connection
database.
openGauss=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

-- Drop user jack.
openGauss=# DROP USER jack;

-- Drop the test_clean_connection database.
openGauss=# DROP DATABASE test_clean_connection;
```

7.13.37 CLOSE

Description

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 } ;
```

Parameters

- **cursor_name**
Specifies the name of a cursor to be closed.
- **ALL**
Closes all open cursors.

Examples

See [Examples](#) in section "FETCH."

Helpful Links

[FETCH](#) and [MOVE](#)

7.13.38 CLUSTER

Description

- Clusters a table based on an index.
- CLUSTER instructs GaussDB to cluster the table specified by the table name based on the index specified by the index name. The index must have been defined by the 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 you use an index to query a table for a range or multiple rows, CLUSTER will also 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. This saves disk accesses and speeds 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 GUC parameter **xc_maintenance_mode** is not set to **on**, the CLUSTER operation 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: name of an existing index.

- **partition_name**

Specifies the partition name.

Value range: an existing partition name.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create a partitioned table.
openGauss=# 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.
openGauss=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

-- Cluster the tpcds.inventory_p1 table.
openGauss=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

-- Cluster the p3 partition.
openGauss=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;

-- Recluster a table.
openGauss=# CLUSTER;

-- Drop the index.
openGauss=# DROP INDEX tpcds.ds_inventory_p1_index1;

-- Drop the partitioned table.
openGauss=# DROP TABLE tpcds.inventory_p1;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.39 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 a system administrator to comment on a SYSADMIN role, or have the CREATEROLE permission to comment on non-SYSADMIN 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 |
  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**
Name of an aggregate function.
- **agg_type**
Data type of the aggregate function parameters.
- **source_type**
Source data type in the type conversion.
- **target_type**
Target data type in the type conversion.

- **object_name**
Object name.
- **table_name.column_name**
view_name.column_name
Column name You can add the table name or view name as the prefix.
- **constraint_name**
Table constraint whose comment is defined or modified.
- **table_name**
Table name.
- **function_name**
Function name.
- **argmode,argname,argtype**
Mode, name, and type of the function parameter.
- **large_object_oid**
OID of a large object.
- **operator_name**
Operator name.
- **left_type,right_type**
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**
Comments.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer table.
openGauss=# CREATE TABLE tpcds.customer
(
  c_customer_sk      INTEGER      NOT NULL,
  c_customer_id      CHAR(16)     NOT NULL
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.customer VALUES (50, 'AAAAAAAAABAAAAAAA'),(100,
'AAAAAAAAACAACAAAAAA'),(150, 'AAAAAAAADAAAAAAA');

-- Create the tpcds.customer_demographics_t2 table.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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.
openGauss=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

-- Drop the view.
openGauss=# DROP VIEW tpcds.customer_details_view_v2;

-- Drop tpcds.customer_demographics_t2.
openGauss=# DROP TABLE tpcds.customer_demographics_t2;

-- Drop the tpcds.customer table.
openGauss=# DROP TABLE tpcds.customer;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.40 COMMIT | END

Description

COMMIT or END commits all operations of a transaction.

Precautions

Only the creator of a transaction or system administrators 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.
openGauss=# CREATE SCHEMA tpcds;

-- Create a table.
openGauss=# 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.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO tpccs.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
openGauss=# INSERT INTO tpccs.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300,
'BAD', 1, 0, 0);

-- Commit the transaction to make all changes permanent.
openGauss=# COMMIT;

-- Query data.
openGauss=# SELECT * FROM tpccs.customer_demographics_t2;

-- Drop the tpccs.customer_demographics_t2 table.
openGauss=# DROP TABLE tpccs.customer_demographics_t2;

-- Drop the schema.
openGauss=# DROP SCHEMA tpccs CASCADE;
```

Helpful Links

[ROLLBACK](#)

7.13.41 COMMIT PREPARED

Description

Commits a prepared two-phase transaction.

Precautions

- The function is only available in maintenance mode (when the 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.
openGauss=# begin;

-- Prepare a transaction whose identifier is trans_test.
openGauss=# PREPARE TRANSACTION 'trans_test';

-- Create a table.
openGauss=# CREATE TABLE item1(id int);

-- Commit the transaction whose identifier is trans_test.
openGauss=# COMMIT PREPARED 'trans_test';

-- Drop the table.
openGauss=# DROP TABLE item1;
```

Helpful Links

[PREPARE TRANSACTION](#) and [ROLLBACK PREPARED](#)

7.13.42 COPY

Description

COPY 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** command. When the **enable_copy_server_files** parameter is enabled, users with the SYSADMIN permission or users who inherit the permissions of the built-in role gs_role_copy_files are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** command. By default, database configuration files and key files are not allowed, and you can run the **COPY FROM FILENAME** or **COPY TO FILENAME** command for certificate files and audit logs to prevent unauthorized users from viewing or modifying sensitive files. When **enable_copy_server_files** is set to **on**, administrators 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.
- **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 into 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.
- 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.

Syntax

- Copy data from a file to a table.

```
COPY table_name [ ( column_name [, ...] ) ]  
  FROM { 'filename' | STDIN }  
  [ [ USING ] DELIMITERS 'delimiters' ]  
  [ WITHOUT ESCAPING ]  
  [ LOG ERRORS ]  
  [ 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

In the syntax, **FIXED FORMATTER ({ column_name(offset, length) } [, ...])** and non-conflicting items of **[copy_option [...]]** can be in any sequence.

- 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

1. The restrictions on the COPY TO syntax are as follows:
(**query**) is incompatible with **[USING] DELIMITERS**. If the data comes from a query result, COPY TO cannot specify **[USING] DELIMITERS**.
2. Use spaces to separate **copy_option** following **FIXED FORMATTER**.
3. **copy_option** is the native parameter, while **option** is the parameter imported by a compatible foreign table.
4. In the syntax, **FIXED FORMATTER ({ column_name(offset, length) } [, ...])** and non-conflicting items of **[copy_option [...]]** can be in any sequence.

The optional parameters of the option clause are as follows:

```
FORMAT 'format_name'  
| OIDS [ boolean ]  
| 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 optional parameters of the copy_option clause are as follows:

```
OIDS  
| 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:

1. The column length of each record is the same.
2. Spaces are used for column padding. Columns of the numeric type are left-aligned and columns of the string type are right-aligned.
3. No delimiters are used between columns.

- **[USING] DELIMITERS 'delimiters'**
The string that separates columns within each row (line) of the file, and it cannot be larger than 10 bytes.
Value range: The delimiter cannot include any of the following characters:
\\abcdefghijklmnopqrstuvwxyz0123456789
Value range: The default value is a tab character in text format and a comma in CSV format.
- **WITHOUT ESCAPING**
Specifies, in 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 DN 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 7-53](#).
 - 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 **LOG ERRORS DATA** parameter.

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.

- **REJECT LIMIT 'limit'**

Used with the **LOG ERROR** option 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 **COPY FROM** 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**

- **OIDS**

Specifies the internal OID to be copied for each row.

 NOTE

An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.

Value range: **true**, **on**, **false**, and **off**.

Default value: **false**

- **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 value of **null**. The delimiter for the CSV format cannot be same as the value of **quote**.
- 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**, **false**, and **off**.

Default value: **false**

– USEEOF

Does not report an error for `\.` in the imported data.

Value range: **true**, **on**, **false**, and **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 marks (""). 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: name of an existing column.

– FORCE_NOT_NULL (column_name [, ...])

In **CSV COPY FROM** mode, the value for a specified column cannot be null.

Value range: name of an existing column.

– ENCODING

Specifies the encoding of data files. If this option is omitted, the current client encoding is used.

– 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**, **false**, and **off**.

- If this parameter is set to **true** or **on** and the number of source data files exceeds the number of foreign table columns, excessive columns will be ignored.
- When the parameter is **false** or **off**, and the number of data source files is more than the number of foreign table columns, the following error information will be displayed:
extra data after last expected column

Default 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**, **false**, and **off**.

- If this parameter is set to **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
- If this parameter is set to **false** or **off** and an error occurs when there are invalid characters, the import will be interrupted.

Default value: **false** or **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** or **on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message like "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**, **false**, and **off**.

Default value: **false** or **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 the Oracle Database 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.

– OIDS

Specifies the internal OID to be copied for each row.

 NOTE

An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.

– 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**
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 [, ...]
In **CSV COPY FROM** mode, the value for a specified column cannot be null.
Value range: name of 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: name of an existing column.
- **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 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 [AS]** 'escape_character'
Specifies an escape character for a CSV file. The value must be a single-byte character.
Default value: double quotation marks (""). 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: .abcdefghijklmnopqrstuvwxyz0123456789.

- **ENCODING 'encoding_name'**
Specifies the name of a file encoding format.
Value range: a valid encoding format.
Default value: current encoding format.
 - **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** or **on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message like "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**, **false**, and **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 CN. Such errors will not be recorded in the COPY error table (enabled using **LOG ERRORS REJECT LIMIT**) but will be reported later by DN. 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 the Oracle Database 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

```
openGauss=> 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.
openGauss=> 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, set the **enable_copy_server_files** parameter to **on**. Then, administrators can use the COPY function. To perform the COPY operation, common users must also join the **gs_role_copy_files** group.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.ship_mode table.
openGauss=# 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.
openGauss=# 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.
openGauss=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

-- Output tpcds.ship_mode to STDOUT.
openGauss=# 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').
openGauss=# 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)).
openGauss=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

-- Create the tpcds.ship_mode_t1 table.
openGauss=# 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.
openGauss=# COPY tpcds.ship_mode_t1 FROM STDIN;
```

```
-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table.
openGauss=# 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.
openGauss=# 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').
openGauss=# 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.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).
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.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;

-- Drop tables and the schema.
openGauss=# DROP TABLE tpcds.ship_mode;
openGauss=# DROP TABLE tpcds.ship_mode_t1;
openGauss=# DROP SCHEMA tpcds;
```

7.13.43 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, the GUC parameter **enable_security_policy** is set to **on**.

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 that complies with the [Identifier Naming Conventions](#).

- **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**, **SHOW**, **LOGIN_ANY**, **LOGIN_FAILURE**, **LOGIN_SUCCESS**, and **LOGOUT**.
- **DML**
Specifies the operations that are audited in the database: **SELECT**, **COPY**, **DEALLOCATE**, **DELETE**, **EXECUTE**, **INSERT**, **PREPARE**, **REINDEX**, **TRUNCATE**, and **UPDATE**.
- **ALL**
Specifies all operations supported by the specified DDL or DML statements 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**
Specifies 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.
openGauss=# CREATE USER dev_audit PASSWORD 'xxxxxxxxxx';
openGauss=# CREATE USER bob_audit password 'xxxxxxxxxx';

-- Create table tb_for_audit.
openGauss=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

-- Create a resource label.
openGauss=# CREATE RESOURCE LABEL adt_lb0 ADD TABLE(tb_for_audit);

-- Perform the CREATE operation on the database to create an audit policy.
openGauss=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- Perform the SELECT operation on the database to create an audit policy.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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');

-- Drop audit policies.
openGauss=# DROP AUDIT POLICY adt1, adt2, adt3, adt4;

-- Drop a resource label.
openGauss=# DROP RESOURCE LABEL adt_lb0;
```

```
-- Drop the tb_for_audit table.  
openGauss=# DROP TABLE tb_for_audit;  
  
-- Drop the dev_audit and bob_audit users.  
openGauss=# DROP USER dev_audit, bob_audit;
```

Helpful Links

[ALTER AUDIT POLICY](#) and [DROP AUDIT POLICY](#)

7.13.44 CREATE BARRIER

Description

Creates a barrier for cluster nodes. The barrier can be used for data restoration.

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**
(Optional) Specifies the name of a barrier.
Value range: a string that complies with the [Identifier Naming Conventions](#).

Examples

```
-- Create a barrier without specifying its name.  
openGauss=# CREATE BARRIER;  
  
-- Specify the barrier name.  
openGauss=# CREATE BARRIER 'barrier1';
```

7.13.45 CREATE CONVERSION

Description

Defines a new conversion between two character set encodings.

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.

```
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;
```

Examples

```
-- Use myfunc to create an encoding conversion from UTF8 to LATIN1.  
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

7.13.46 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 [=] 'compatibilty_type' ] |
  [ TABLESPACE [=] tablespace_name ] |
  [ CONNECTION LIMIT [=] connlimit ] }[...];
```

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 7-110 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	-
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	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
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	ISO8859 1
LATIN2	ISO 8859-2, ECMA 94	Central European languages	Yes	Yes	1	ISO8859 2
LATIN3	ISO 8859-3, ECMA 94	South European languages	Yes	Yes	1	ISO8859 3
LATIN4	ISO 8859-4, ECMA 94	North European languages	Yes	Yes	1	ISO8859 4
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	Yes	1	ISO8859 9

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
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
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	No	1-2	-
SQL_ASCII	Unspecified (see the text)	Any	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

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
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	-
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

NOTICE

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 GUC parameter **client_encoding**.)
-

- **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.

- **DBCOMPATIBILITY [=] 'compatibility_type'**

Specifies the compatible database type. The default value is **MySQL**.

Value range: **MYSQL**, **TD**, **ORA**, and **PG**. They represent MySQL-, TD-, Oracle-, and PG-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.
- **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 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.

Examples

```
-- Create users jim and tom.
openGauss=# CREATE USER jim PASSWORD '*****';
openGauss=# CREATE USER tom PASSWORD '*****';

-- Create database music using GBK (the local encoding type is also GBK).
openGauss=# CREATE DATABASE music ENCODING 'GBK' template = template0;

-- Create database music2 and specify user jim as its owner.
openGauss=# CREATE DATABASE music2 OWNER jim;

-- Create database music3 using template template0 and specify user jim as its owner.
openGauss=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

-- Set the maximum number of connections to database music to 10.
openGauss=# ALTER DATABASE music CONNECTION LIMIT= 10;

-- Rename database music to music4.
openGauss=# ALTER DATABASE music RENAME TO music4;

-- Change the owner of database music2 to user tom.
openGauss=# ALTER DATABASE music2 OWNER TO tom;

-- Drop databases.
openGauss=# DROP DATABASE music2;
openGauss=# DROP DATABASE music3;
openGauss=# DROP DATABASE music4;

-- Drop the jim and tom users.
openGauss=# DROP USER jim;
openGauss=# DROP USER tom;

-- Create a database compatible with the TD format.
openGauss=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'TD';

-- Create a database compatible with the ORA format.
openGauss=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA';

-- Drop the databases that are compatible with the TD and ORA formats.
openGauss=# DROP DATABASE td_compatible_db;
openGauss=# 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 setting of **Encoding** of the new database does not match that of the template database (SQL_ASCII) ('**GBK**', '**UTF8**', '**LATIN1**', or '**GB18030**'), **template [=] template0** must be specified.

7.13.47 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 path is a symbolic link.
- 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.  
openGauss=# CREATE OR REPLACE DIRECTORY dir as '/tmp/';
```

Helpful Links

[ALTER DIRECTORY](#) and [DROP DIRECTORY](#)

7.13.48 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 SELECT statement uses the default values of the output parameters when calling the function. When the CALL statement calls the function, it requires that the output parameters must be specified. When the CALL statement calls an overloaded PACKAGE function, it can use the default values of the output parameters. For details, see examples in [CALL](#).
- Only the functions compatible with PG or those with the PACKAGE attribute can be overloaded. 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 can use the SELECT statement to specify different parameters using identical functions, but cannot use the CALL statement to call identical functions without the PACKAGE attribute.
- 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. When creating a function, the user can revoke the default execution permissions from **PUBLIC** and grant them to other 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 calling functions without parameters inside another function, you can omit brackets and call functions using their names directly.
- When functions with output parameters are called inside another function which is an assignment expression, you can omit the output parameters of the called functions.
- 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**'.
- Formal parameters cannot be overloaded if only the custom ref cursor type is different from the sys_refcursor type.
- Functions with the **OUT** parameter do not support nested calling. For example, you are advised to change **b := func(a,func(c,1));** to **tmp := func(c,1); b := func(a,tmp).**
- 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.

Syntax

- Syntax (compatible with PG) 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 }
    | { PACKAGE }

    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { { TO | = } value | FROM CURRENT } }
][...]
{
    AS 'definition'
    | AS 'obj_file', 'link_symbol'
}
```

- Oracle syntax of 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}
    | {PACKAGE}
    | {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 create (optionally schema-qualified).
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **argname**
Specifies the parameter name of the function.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **argmode**
Specifies the parameter mode of the function.
Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameters of **OUT** 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 parameter type of the function. You can use **%ROWTYPE** to indirectly reference the type of a table, or **%TYPE** to indirectly reference the type of a column in a table or composite type.
- **expression**
Specifies the default expression of a parameter.

NOTE

It is recommended that you define all default parameters after all non-default parameters.

- **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 exists, the result type of the clause must be the same as that of the output parameter. If there are multiple output parameters, the result type of the clause is **RECORD**. Otherwise, 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.
- **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 O-style database is created, the language used is PL/SQL.
- **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 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**.

NOTICE

The function or stored procedure of the SHIPPABLE/IMMUTABLE type cannot contain EXCEPTION or call functions or stored procedures that contain EXCEPTION.

- **PACKAGE**

Specifies whether the function can be overloaded.

- All PACKAGE and non-PACKAGE functions cannot be overloaded or replaced.
- PACKAGE functions do not support parameters of the VARIADIC type.
- The PACKAGE attribute of functions cannot be modified.

- **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 a function always returns **NULL** when the value of 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: 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** disables the setting.

Value range: a string.

- **DEFAULT**
- **OFF**
- **RESET**
- Specified default value

- **FROM CURRENT**

Uses the value of **configuration_parameter** of the current session.

- **obj_file, link_symbol**

obj_file specifies the absolute path of the dynamic library. **link_symbol** specifies the link symbol of the function, that is, the function name in the C code.

- **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

```
-- Define a function as SQL query.
openGauss=# 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.
openGauss=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
BEGIN
    RETURN i + 1;
END;
$$ LANGUAGE plpgsql;

-- Return the RECORD type.
openGauss=# CREATE OR REPLACE FUNCTION compute(i int, out result_1 bigint, out result_2 bigint)
RETURNS SETOF RECORD
AS $$
BEGIN
    result_1 = i + 1;
    result_2 = i * 10;
RETURN next;
END;
$$LANGUAGE plpgsql;

-- Return a record containing multiple output parameters.
openGauss=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
LANGUAGE SQL;

openGauss=# 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).
openGauss=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Create an overloaded function with the PACKAGE attribute:
openGauss=# CREATE OR REPLACE FUNCTION package_func_overload(col int, col2 int)
RETURN integer package
AS
DECLARE
    col_type text;
BEGIN
    col := 122;
    db_output.print_line('two int parameters ' || col2);
    return 0;
END;
/

openGauss=# CREATE OR REPLACE FUNCTION package_func_overload(col int, col2 smallint)
RETURN integer package
```

```
AS
DECLARE
    col_type text;
BEGIN
    col := 122;
    db_output.print_line('two smallint parameters ' || col2);
    RETURN 0;
END;
/

-- Alter the execution rule of function add to IMMUTABLE (that is, the same result is returned if the
parameter remains unchanged).
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

-- Alter the name of function add to add_two_number.
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

-- Change the owner of function add to omm.
openGauss=# ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO omm;

-- Drop functions.
openGauss=# DROP FUNCTION add_two_number;
openGauss=# DROP FUNCTION func_increment_sql;
openGauss=# DROP FUNCTION func_dup_sql;
openGauss=# DROP FUNCTION func_increment_plsql;
openGauss=# DROP FUNCTION func_add_sql;
```

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.

7.13.49 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 "CREATE ROLE."

Examples

```
-- Create group my_group.  
openGauss=# CREATE GROUP my_group PASSWORD '*****';  
  
-- Drop the group.  
openGauss=# DROP GROUP my_group;
```

Helpful Links

[ALTER GROUP](#), [DROP GROUP](#), and [CREATE ROLE](#)

7.13.50 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 temporary tables or global temporary tables.
- Fast-refresh materialized views support only simple filter queries and UNION ALL queries of base tables.

- Distribution keys cannot be specified when an incremental MV is created.
- After a fast-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- IUD operations 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.

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 that complies with the [Identifier Naming Conventions](#).
- **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 that complies with the [Identifier Naming Conventions](#).
- **TABLESPACE tablespace_name**
Specifies a tablespace to which the new materialized view belongs. If it 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.  
openGauss=# CREATE TABLE my_table (c1 int, c2 int);  
  
-- Create a fast-refresh materialized view.  
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;  
  
-- Write data to the base table.  
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);  
  
-- Fast refresh the fast-refresh materialized view my_imv.  
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;  
  
-- Drop a fast-refresh materialized view.  
openGauss=# DROP MATERIALIZED VIEW my_imv;  
  
-- Drop the ordinary table my_table.  
openGauss=# 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](#)

7.13.51 CREATE INDEX

Description

Creates an index in a specified table.

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

- 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.
- When creating a unique index on a partitioned table, ensure that the index contains distribution keys. If an index does not contain a partition key, only global partitioned indexes can be created.
- A user granted 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.

Syntax

- Create an index on a table.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ schemaname.]index_name ] ON table_name  
[ USING method ]
```

```
( { { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS  
{ FIRST | LAST } ] } [, ...] )  
[ WITH ( { storage_parameter = value } [, ...] ) ]  
[ TABLESPACE tablespace_name ]  
[ WHERE predicate ];
```

- Create an index on a partitioned table.

```
CREATE [ UNIQUE ] INDEX [ [schemaname.]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 ]  
[ WITH ( { storage_parameter = value } [, ...] ) ]  
[ TABLESPACE tablespace_name ]  
[ WHERE predicate ];
```

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.
- Partitioned tables do not support index creation by using CONCURRENTLY. 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 the query, 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.
- 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, 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. For 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. 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 create. No schema name can be included here; the index is always created in the same schema as its parent table.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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**: The B-tree index uses a structure that is similar to the B+ tree structure to store data key values, facilitating index search. B-tree indexes support comparison queries with ranges specified. When an index is created in a Ustore table, the index is automatically changed to UB-tree. Row-store tables support the **btree** index type (default).
- **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 composite 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.

- **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:
Indexes except for psort support the **FILLFACTOR** parameter.
 - **FILLFACTOR**
The fill factor of an index is a percentage from 10 to 100.
Value range: 10–100.
 - **CROSSBUCKET**
Specifies whether cross-hash bucket indexes are used. Only B-tree indexes are supported.
Value range: **ON** and **OFF**
Default: **ON**
- **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, the **WHERE** clause with a **UNIQUE** constraint can be used to enforce uniqueness over a subset of 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 that complies with the [Identifier Naming Conventions](#).

- **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.
openGauss=# CREATE SCHEMA tpcds;
openGauss=# 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.
openGauss=# 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.
openGauss=# 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:
openGauss=# 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.
openGauss=# 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.
openGauss=# CREATE INDEX CONCURRENTLY ds_ship_mode_t1_index4 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

-- Rename an existing index.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

-- Set the index as unusable.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

-- Rebuild an index.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

-- Drop an existing index.
openGauss=# DROP INDEX tpcds.ds_ship_mode_t1_index2;
```

```
-- Drop the table.
openGauss=# DROP TABLE tpcds.ship_mode_t1;

-- Create a tablespace.
openGauss=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
openGauss=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
openGauss=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
openGauss=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
-- Create the tpcds.customer_address_p1 table.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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;

-- Change the tablespace of the partitioned table index CA_ADDRESS_SK_index2 to example1.
openGauss=# 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.
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION
CA_ADDRESS_SK_index3 TABLESPACE example2;

-- Rename a partitioned table index.
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION
CA_ADDRESS_SK_index1 TO CA_ADDRESS_SK_index4;

-- Drop the indexes and the partitioned table.
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index1;
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index2;
openGauss=# DROP TABLE tpcds.customer_address_p1;
-- Drop tablespaces.
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
```

```
openGauss=# DROP TABLESPACE example3;  
openGauss=# 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 DISTINCTConstraints:
 - 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 cannot exceed one-third of the page size.
 - Partial indexes cannot be created in a partitioned table.
 - When creating a unique index on a partitioned table, ensure that the index contains distribution keys. If an index does not contain a partition key, only global partitioned indexes can be created.

7.13.52 CREATE LANGUAGE

Function

Defines a new procedural language.

Precautions

The current version does not support this syntax.

7.13.53 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, the GUC parameter **enable_security_policy** is set to **on**.

Syntax

```
CREATE MASKING POLICY policy_name masking_clause[, ...] [ policy_filter_clause ] [ENABLE | DISABLE];
```

- **masking_clause**
masking_function ON LABEL(label_name[, ...])
- **masking_function**
The maskall function is not preset. It is hard-coded and cannot be displayed by running **\df**.
The masking methods during presetting are as follows:
{ maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking | shufflemasking | alldigitsmasking | 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 that complies with the [Identifier Naming Conventions](#).
- **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**
Specifies 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.  
openGauss=# CREATE USER dev_mask PASSWORD 'xxxxxxxxxx';  
openGauss=# CREATE USER bob_mask PASSWORD 'xxxxxxxxxx';  
  
-- Create table tb_for_masking.
```

```
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

-- Create a resource label for sensitive column col1.
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a resource label for sensitive column col2.
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a masking policy for the operation of accessing sensitive column col1.
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- 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.
openGauss=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_lb2) FILTER ON
ROLES(dev_mask, bob_mask), APP(gsq), IP('10.20.30.40', '127.0.0.0/24');

-- Drop masking policies.
openGauss=# DROP MASKING POLICY maskpol1, maskpol2;

-- Drop resource labels.
openGauss=# DROP RESOURCE LABEL mask_lb1, mask_lb2;

-- Drop the tb_for_masking table.
openGauss=# DROP TABLE tb_for_masking;

-- Drop the dev_mask and bob_mask users.
openGauss=# DROP USER dev_mask, bob_mask;
```

Helpful Links

[5.1.13.14-ALTER MASKING POLICY](#) and [5.1.13.14.96-DROP MASKING POLICY](#)

7.13.54 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.

Syntax

```
CREATE MATERIALIZED VIEW mv_name
[ (column_name [, ...] ) ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
```



```
[ TABLESPACE tablespace_name ]  
AS query;
```

Parameters

- **mv_name**
Name (optionally schema-qualified) of the materialized view to be created.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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 that complies with the [Identifier Naming Conventions](#).
- **WITH (storage_parameter [= value] [, ...])**
Specifies an optional storage parameter for a table or an index. For details, see [CREATE TABLE](#).
- **TABLESPACE tablespace_name**
Specifies a tablespace to which the new materialized view belongs. If it is not specified, the default tablespace is used.
- **AS query**
Specifies the **SELECT**, **TABLE**, or **VALUES** command. This query will be run in a security-constrained operation.

Examples

```
-- Create an ordinary table.  
openGauss=# CREATE TABLE my_table (c1 int, c2 int);  
  
-- Create a complete-refresh materialized view.  
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;  
  
-- Write data to the base table.  
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);  
  
-- Completely refresh the complete-refresh materialized view my_mv.  
openGauss=# REFRESH MATERIALIZED VIEW my_mv;  
  
-- Drop a complete-refresh materialized view.  
openGauss=# DROP MATERIALIZED VIEW my_mv;  
  
-- Drop the ordinary table my_table.  
openGauss=# 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](#)

7.13.55 CREATE NODE

Description

Creates a cluster node.

Precautions

CREATE NODE is an API of the cluster management tool. You are advised not to use this API, because doing so affects the cluster. Only an administrator has the permission to use this API.

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 that complies with the [Identifier Naming Conventions](#).
- **TYPE = nodetype**
Specifies the type of a node.
Value range:
 - 'coordinator'
 - 'datanode'
- **HOST = hostname**
Specifies the name or IP address of the primary node corresponding to the specified node.
- **PORT = portnum**
Specifies the port number of the primary node to which the specified node is bound.
- **HOST1 = hostname**
Specifies the name or IP address of the standby node corresponding to the specified node.
- **PORT1 = portnum**
Specifies the port number of the standby node 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 node to listen on the control transmission channel. The TCP protocol is used to listen on connections.
- **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 node to listen on the control channel. It is a TCP port.

Helpful Links

[ALTER NODE](#) and [DROP NODE](#)

7.13.56 CREATE NODE GROUP

Description

Creates a cluster node group.

Precautions

- CREATE NODE GROUP is an API 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 that complies with the [Identifier Naming Conventions](#).
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 that complies with the [Identifier Naming Conventions](#).
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](#)

7.13.57 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.
- If a SELECT statement calls a stored procedure with output parameters, output parameters cannot be specified. When a CALL statement calls a stored procedure involving a non-overloaded function, output parameters must be specified. When a CALL statement calls an overloaded PACKAGE function, the output parameters may not be specified. For details, see examples in [CALL](#).
- A stored procedure with the PACKAGE attribute can use overloaded functions.
- When you create a procedure, you cannot nest average functions within aggregate or other functions.
- 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 functions with output parameters are called in a stored procedure which is an assignment expression, you can omit the output parameters of the called functions.
- 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**'.
- 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.

Syntax

```
openGauss=# CREATE [ OR REPLACE ] PROCEDURE procedure_name
[ ( ( [ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ] ) [, ... ] ) ]
[
    { IMMUTABLE | STABLE | VOLATILE }
    | { SHIPPABLE | NOT SHIPPABLE }
    | { PACKAGE }
    | [ 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 that complies with the [Identifier Naming Conventions](#).
- **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 parameters of **OUT** 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 that complies with the [Identifier Naming Conventions](#).
 - **argtype**
Specifies the type of an argument. You can use **%ROWTYPE** to indirectly reference the type of a table, or **%TYPE** to indirectly reference the type of a column in a table or composite type.
Value range: a valid data type.
 - **expression**
Specifies the default expression of a parameter.
-  **NOTE**
- It is recommended that you define all default parameters after all non-default parameters.
- **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 **argname**. The following order is advised:
argname, **argmode**, and **argtype**.

Examples

```
-- Create a stored procedure.
openGauss=# 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.
openGauss=# SELECT prc_add(2,3);

-- Create a stored procedure whose parameter type is VARIADIC.
openGauss=# CREATE OR REPLACE PROCEDURE pro_variadic (var1 VARCHAR2(10) DEFAULT 'hello!',var4
VARIADIC int4[])
AS
BEGIN
    db_output.print_line(var1);
END;
/

-- Execute the stored procedure.
openGauss=# 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.
openGauss=# CREATE TABLE tb1(a integer);
openGauss=# CREATE PROCEDURE insert_data(v integer)
SECURITY INVOKER
AS
BEGIN
    INSERT INTO tb1 VALUES(v);
END;
/

-- Call the stored procedure.
openGauss=# CALL insert_data(1);

-- Create a stored procedure with the PACKAGE attribute.
openGauss=# create or replace procedure package_func_overload(col int, col2 out varchar)
package
as
declare
    col_type text;
begin
    col2 := '122';
    db_output.print_line('two varchar parameters ' || col2);
end;
/

-- Drop stored procedures.
openGauss=# DROP PROCEDURE prc_add;
openGauss=# DROP PROCEDURE pro_variadic;
openGauss=# DROP PROCEDURE insert_data;
openGauss=# DROP PROCEDURE package_func_overload;
```

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.

7.13.58 CREATE RESOURCE LABEL

Description

Creates a resource label.

Precautions

Only users 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 that complies with the [Identifier Naming Conventions](#).
- **resource_type**
Specifies the type of database resources to be labeled.
- **resource_path**
Specifies the path of database resources.

Examples

```
-- Create table tb_for_label.
openGauss=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create schema schema_for_label.
openGauss=# CREATE SCHEMA schema_for_label;

-- Create view view_for_label.
openGauss=# CREATE VIEW view_for_label AS SELECT 1;

-- Create function func_for_label.
openGauss=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;
```



```
-- Create a resource label based on a table.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create a resource label based on columns.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add
COLUMN(public.tb_for_label.col1);

-- Create a resource label based on a schema.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create a resource label based on a view.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create a resource label based on a function.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Drop resource labels.
openGauss=# DROP RESOURCE LABEL func_label, view_label, schema_label, column_label, table_label;

-- Drop the func_for_label function.
openGauss=# DROP FUNCTION func_for_label;

-- Drop the view_for_label view.
openGauss=# DROP VIEW view_for_label;

-- Drop the schema_for_label schema.
openGauss=# DROP SCHEMA schema_for_label;

-- Drop the tb_for_label table.
openGauss=# DROP TABLE tb_for_label;
```

Helpful Links

[5.1.13.14.17-ALTER RESOURCE LABEL](#) and [5.1.13.14.102-DROP RESOURCE LABEL](#)

7.13.59 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 option clauses for configuring role information 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'  
| 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 that follows the [Identifier Naming Conventions](#) with a maximum of 63 characters. If a 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.

The 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, 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 role, enclose the user password in single quotation marks.

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 SYSADMIN, which has the highest permission. The default value is **NOSYSADMIN**.

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.
The default value is **NOMONADMIN**.

- **OPRADMIN | NOOPRADMIN**

Specifies whether a role is an OPRADMIN.
The default value is **NOOPRADMIN**.

- **POLADMIN | NOPOLADMIN**

Specifies whether a role is a POLADMIN.
The default value is **NOPOLADMIN**.

- **AUDITADMIN | NOAUDITADMIN**

Specifies whether a role has the audit and management attributes.
The default value is **NOAUDITADMIN**.

- **CREATEDB | NOCREATEDB**

Specifies a role's permission to create databases.
A new role does not have the permission to create databases.
The default value is **NOCREATEDB**.

- **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.

The default value is **NOCREATEROLE**.

- 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 with the **LOGIN** attribute can be considered as a user.

The default value is **NOLOGIN**.

- **REPLICATION | NOREPLICATION**

Specifies whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role with **REPLICATION** attribute is specific to replication.

The default value is **NOREPLICATION**.

- **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.
- **connlimit** is calculated for 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 **-1** means no limit.

- **VALID BEGIN**

Sets a date and time when the role's password takes effect. If this clause is omitted, the password takes effect immediately.

- **VALID UNTIL**

Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the password will be valid for all time.

- **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 *****.
openGauss=# CREATE ROLE manager IDENTIFIED BY '*****';

-- Create a role with its validity from January 1, 2015 to January 1, 2026.
openGauss=# CREATE ROLE miriam WITH LOGIN PASSWORD '*****' VALID BEGIN '2015-01-01' VALID
UNTIL '2026-01-01';

-- Change the password of role manager to *****.
openGauss=# ALTER ROLE manager IDENTIFIED BY '*****' REPLACE '*****';

-- Change role manager to SYSADMIN.
openGauss=# ALTER ROLE manager SYSADMIN;

-- Drop role manager.
openGauss=# DROP ROLE manager;

-- Drop role miriam.
openGauss=# DROP GROUP miriam;
```

Helpful Links

[SET ROLE](#), [ALTER ROLE](#), [DROP ROLE](#), [GRANT](#), and [REVOKE](#)

7.13.60 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 who 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 policy for row-level security. The conditions of the permissive policy are joined through the OR expression.
RESTRICTIVE enables the restrictive policy for row-level security. 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 limited 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**, **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 7-111 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

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

Specifies database users limited by a row-level security policy.

CURRENT_USER indicates the username in the current operating 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 limited by the row-level security policy.

- **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.
openGauss=# CREATE USER alice PASSWORD 'xxxxxxxxx';

-- Create user bob.
openGauss=# CREATE USER bob PASSWORD 'xxxxxxxxx';

-- Create data table all_data.
openGauss=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

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

-- Grant the read permission on the all_data table to users alice and bob.
openGauss=# GRANT SELECT ON all_data TO alice, bob;
```



```
-- Enable the row-level security policy.
openGauss=# 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.
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);

-- View information about the all_data table.
openGauss=# \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_rls"
    USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, enable_rowsecurity=true

-- Run SELECT.
openGauss=# SELECT * FROM all_data;
id | role | data
---+---+---
1 | alice | alice data
2 | bob   | bob data
3 | peter | peter data
(3 rows)

-- Grant the login permission to the user.
openGauss=# ALTER USER alice LOGIN;

openGauss=# 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.
openGauss=# SELECT * FROM all_data;
id | role | data
---+---+---
1 | alice | alice data
(1 row)

openGauss=# 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)

-- Drop a row-level security policy.
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

-- Drop the all_data table.
openGauss=# DROP TABLE public.all_data;

-- Drop users alice and bob.
openGauss=# DROP USER alice, bob;
```

Helpful Links

[DROP ROW LEVEL SECURITY POLICY](#)

7.13.61 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 a 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 name must be unique.

The schema name cannot start with **pg_**.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **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.
openGauss=# CREATE ROLE role1 IDENTIFIED BY '*****1';

-- Create a schema named role1 for the role1 role. The owner of the films and winners tables created by
the clause is role1.
openGauss=# 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;

-- Drop the schema.
openGauss=# DROP SCHEMA role1 CASCADE;
-- Drop the user.
openGauss=# DROP USER role1 CASCADE;
```

Helpful Links

[ALTER SCHEMA](#) and [DROP SCHEMA](#)

7.13.62 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 string 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.
- **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 $-2^{63}-1$. **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 $2^{63}-1$, 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.

```
openGauss=# CREATE SEQUENCE serial
START 101
CACHE 20;
```

Select the next number from the sequence.

```
openGauss=# SELECT nextval('serial');
nextval
-----
101
```

Select the next number from the sequence.

```
openGauss=# SELECT nextval('serial');
nextval
-----
102
```

Create a sequence associated with the table.

```
openGauss=# 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)    )
```

```
);  
openGauss=# CREATE SEQUENCE serial1  
START 101  
CACHE 20  
OWNED BY customer_address.ca_address_sk;  
-- Delete the sequence.  
openGauss=# DROP TABLE customer_address;  
openGauss=# DROP SEQUENCE serial cascade;  
openGauss=# DROP SEQUENCE serial1 cascade;
```

Helpful Links

[DROP SEQUENCE](#) and [ALTER SEQUENCE](#)

7.13.63 CREATE SERVER

Description

Creates a foreign server.

A foreign server stores OBS server information or 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** 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

```
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**.

- **OPTIONS ({ option_name ' value ' } [, ...])**

Specifies the parameters for the foreign server. The detailed parameter description is as follows:

- **encrypt**

Specifies whether data is encrypted. This parameter is available only when **type** is **OBS**. The default value is **on**.

Value range:

- **on** indicates that data is encrypted and HTTPS is used for communication.
 - **off** indicates that data is not encrypted and HTTP is used for communication.

- **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, the AK value is encrypted and saved to the metadata table of the database. 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. When you create a foreign table, the SK value is encrypted and saved to the metadata table of the database. This parameter is available only when **type** is set to **OBS**.

Examples

Create **my_server**, in which **file_fdw** is the built-in foreign data wrapper.

```
-- Create my_server.
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;

-- Delete my_server.
openGauss=# 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.
openGauss=# 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 'xxxxxxx'
);

-- Delete the server.
openGauss=# DROP SERVER server_remote;
```

Helpful Links

[ALTER SERVER](#) and [DROP SERVER](#)

7.13.64 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, 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. In addition, you need to run the **DROP SYNONYM** command before the current session ends.
- After an object is deleted, the synonym associated with the object will not be deleted in cascading mode. If you continue to access the synonym, an error message is displayed, indicating that the synonym has expired.

Syntax

```
CREATE [ OR REPLACE ] SYNONYM synonym_name  
FOR object_name;
```

Parameters

- **synonym**
Specifies the name of the synonym to be created, which can contain the schema name.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **object_name**
Specifies the name of an object that is associated (optionally with schema names).
Value range: a string that complies with the [Identifier Naming Conventions](#).

NOTE

object_name can be the name of an object that does not exist.

 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.
openGauss=# CREATE SCHEMA ot;

-- Create table ot.t1 and its synonym t1.
openGauss=# CREATE TABLE ot.t1(id int, name varchar2(10)) DISTRIBUTE BY hash(id);
openGauss=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

-- Use synonym t1.
openGauss=# SELECT * FROM t1;
openGauss=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
openGauss=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;

-- Create synonym v1 and its associated view ot.v_t1.
openGauss=# CREATE SYNONYM v1 FOR ot.v_t1;
openGauss=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

-- Use synonym v1.
openGauss=# SELECT * FROM v1;

-- Create overloaded function ot.add and its synonym add.
openGauss=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

openGauss=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS
decimal(5,2) AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

openGauss=# CREATE OR REPLACE SYNONYM add FOR ot.add;

-- Use synonym add.
openGauss=# SELECT add(1,2);
openGauss=# SELECT add(1.2,2.3);

-- Create stored procedure ot.register and its synonym register.
openGauss=# 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;
/

openGauss=# CREATE OR REPLACE SYNONYM register FOR ot.register;

-- Use synonym register to call the stored procedure.
openGauss=# CALL register(3,'mia');

-- Drop synonyms.
openGauss=# DROP SYNONYM t1;
openGauss=# DROP SYNONYM IF EXISTS v1;
openGauss=# DROP SYNONYM IF EXISTS add;
```

```
openGauss=# DROP SYNONYM register;  
openGauss=# DROP SCHEMA ot CASCADE;
```

Helpful Links

[ALTER SYNONYM](#) and [DROP SYNONYM](#)

7.13.65 CREATE TABLE

Description

Creates an initially empty table in the current database. The table will be owned by the creator.

Precautions

- The primary key constraint and unique constraint in the table must contain distribution keys.
- Distribution keys 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.
- According to the concurrency control policy, if the **DROP TABLE IF EXIST** and **CREATE IF EXIST** statements are performed on the same table concurrently, one of the two will be rolled back.
- 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.

NOTICE

If you create tables infinitely in GaussDB, CNs may be affected as follows:

- Resource exhaustion: Each table occupies certain disk space. Creating tables infinitely 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: Infinite 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 tables infinitely 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 tables in the database properly to avoid infinite table creation and ensure system stability, reliability, and security.

Syntax

- Create a table.

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] [ UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
    ({ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option [ ... ] ] }
    [ ... ])
[ WITH ( {storage_parameter = value} [ , ... ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
[ 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 |
  UNIQUE [ index_parameters ] |
  PRIMARY KEY [ index_parameters ] |
  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 ] |
  PARTIAL CLUSTER KEY ( column_name [ , ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **like_option** is as follows:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |  
PARTITION | REOPTIONS | DISTRIBUTION | ALL }
```

- Range distribution rules

slice_less_than_item is as follows:

```
SLICE slice_name VALUES LESS THAN ({ literal | MAXVALUE } [, ...])  
[ DATANODE dn_name ]
```

slice_start_end_item is as follows:

```
SLICE slice_name_prefix {  
{ START ( literal ) END ( literal ) EVERY ( literal ) } |  
{ START ( literal ) END ( { literal | MAXVALUE } ) } |  
{ START ( literal ) } |  
{ END ( { literal | MAXVALUE } ) }  
}
```

- The LIST distribution rule **slice_values_item** is as follows:

```
SLICE slice_name VALUES (list_values_item) [DATANODE dn_name]
```

list_values_item is as follows:

```
{ DEFAULT | { partition_values_list [, ...] } }
```

partition_values_list is as follows:

```
{ (literal [, ...]) }
```

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, the data in the unlogged table is cleared after a conflict occurs, the OS is restarted, the database is restarted, a switchover is performed, the power supply is cut off, or the database is restarted abnormally, which may cause data loss. 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 an abnormal shutdown or other unexpected operations, 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.
 - **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 copied 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 for a table can contain **OIDS=TRUE** or **OIDS** to specify that each row in the new table is assigned an OID. If **OIDS=FALSE** is specified, no OID is assigned.

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. 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 (row-store) of table data. This parameter cannot be modified once it is set.

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**.

- segment

The data is stored in segment-page mode. This parameter supports only row-store tables. Temporary tables and unlogged tables are not supported. Ustore is not supported.

Value range: **on** and **off**

Default value: **off**

- hashbucket

Creates a hash table that uses buckets. This parameter supports only row-store tables, including row-store range tables.

Value range: **on** and **off**

Default value: **off**

NOTICE

- In current version, DDL operations on hash bucket tables are affected. Therefore, you are advised not to frequently perform DDL operations on hash bucket tables.
- Hash bucket tables are bound to segmented paging storage, that is, when **hashbucket** is set to **on**, **segment** is set to **on**.

- bucketcnt

Specifies the number of buckets of a bucket table when the table is created. The value of this parameter must correspond to a child node group.

The value ranges from 32 to 16384 and must be an integer power of 2.

Default value: **16384**.

- 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. The value **0** indicates that this function 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**

- **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.

- **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 nodes.

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

- When **DISTRIBUTE BY { HASH | RANGE | LIST } (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.
- 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 first column is smaller than the boundary value of the current column in the local shard, values are directly inserted.
 3. If the value of the first column is equal to the boundary value of the current column in the local shard, compare the value of the second column with the boundary value of the next column in the local shard. If the value of the second column is smaller than the boundary value of the next column in the local shard, values are directly inserted. If they are still equal, continue to the comparison until the value of the column is smaller than the boundary value of the column in the local shard, and then insert the values.
 4. If the values of the all columns are greater than the boundary value of the current column in the local shard, compare the value with that in the next shard.

Default value: **HASH(column_name)**. Set **column_name** to the primary key (if any) of the table or the column whose first data type supports distribution keys.

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**

 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.

```
openGauss=# 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. Re-create 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 that 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 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.

 NOTE

If **DISTRIBUTE BY REPLICATION** is not specified, the column table that contains only unique values must contain distribution keys.

- **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.

NOTE

If **DISTRIBUTE BY REPLICATION** is not specified, the column set with a primary key constraint must contain distribution keys.

- **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.

Examples

```
-- Create a simple table.
openGauss=# 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)
);

openGauss=# 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.
openGauss=# CREATE TABLE tpcds.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.
openGauss=# CREATE TABLE tpcds.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%.
openGauss=# CREATE TABLE tpcds.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:
openGauss=# CREATE TABLE tpcds.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.
openGauss=# CREATE UNLOGGED TABLE tpcds.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.
openGauss=# 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.
openGauss=# 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).
openGauss=# 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.
openGauss=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';
-- Specify a tablespace when creating a table.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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:
openGauss=# 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:
openGauss=# 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.
openGauss=# 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.
openGauss=# 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)
);

openGauss=# 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.
openGauss=# 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.
```

```
openGauss=# ALTER TABLE tpccs.warehouse_t21 SET (primarynode=on);
```

Check whether the option is enabled. (The content displayed in **Options** varies according to the version.)

```
openGauss=# \d+ tpccs.warehouse_t21
```

Table "tpccs.warehouse_t21"					
Column	Type	Modifiers	Storage	Stats target	Description
w_warehouse_sk	integer	not null	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: primarynode=on

-- Define a table using HASH distribution.

```
openGauss=# CREATE TABLE tpccs.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);
```

-- View DN information.

```
openGauss=# select node_name from pgxc_node;
 node_name
```

```
-----
coordinator1
datanode1
datanode2
datanode3
datanode4
datanode5
datanode6
(7 rows)
```

-- Define a table using RANGE distribution.

```
openGauss=# CREATE TABLE tpccs.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 datanode1,
  SLICE s2 VALUES LESS THAN (20) DATANODE datanode2,
  SLICE s3 VALUES LESS THAN (30) DATANODE datanode3,
  SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE datanode4
);

-- Example of a multi-column range partitioning policy
openGauss=# 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
);
openGauss=# insert into t_ran1 values(9,5,'a');
openGauss=# insert into t_ran1 values(9,20,'a');
openGauss=# insert into t_ran1 values(9,21,'a');
openGauss=# insert into t_ran1 values(10,5,'a');
openGauss=# insert into t_ran1 values(10,15,'a');
openGauss=# insert into t_ran1 values(10,20,'a');
openGauss=# insert into t_ran1 values(10,21,'a');
openGauss=# insert into t_ran1 values(11,5,'a');
openGauss=# insert into t_ran1 values(11,20,'a');
openGauss=# insert into t_ran1 values(11,21,'a');
openGauss=# 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)
openGauss=# 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.
openGauss=# CREATE TABLE tpceds.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.
openGauss=# CREATE TABLE tpcds.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 datanode1,
SLICE s2 VALUES ('CANADA') DATANODE datanode2,
SLICE s3 VALUES ('UK') DATANODE datanode3,
SLICE s4 VALUES (DEFAULT) DATANODE datanode4
);
-- Add a varchar column to the tpcds.warehouse_t19 table.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

-- Add a check constraint to the tpcds.warehouse_t19 table.
openGauss=# 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.
openGauss=# 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.
openGauss=# 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.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

-- Remove NOT NULL constraints from an existing column.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;

-- Move a table to another tablespace.
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
-- Create the joe schema.
openGauss=# CREATE SCHEMA joe;

-- Move a table to another schema.
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

-- Rename an existing table.
openGauss=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

-- Drop a column from the warehouse_t23 table.
openGauss=# ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;

-- Drop the tablespace, schema joe, and schema tables warehouse.
openGauss=# DROP TABLE tpcds.warehouse_t1;
```

```
openGauss=# DROP TABLE tpcds.warehouse_t2;
openGauss=# DROP TABLE tpcds.warehouse_t3;
openGauss=# DROP TABLE tpcds.warehouse_t4;
openGauss=# DROP TABLE tpcds.warehouse_t5;
openGauss=# DROP TABLE tpcds.warehouse_t6;
openGauss=# DROP TABLE tpcds.warehouse_t7;
openGauss=# DROP TABLE tpcds.warehouse_t8;
openGauss=# DROP TABLE tpcds.warehouse_t9;
openGauss=# DROP TABLE tpcds.warehouse_t10;
openGauss=# DROP TABLE tpcds.warehouse_t11;
openGauss=# DROP TABLE tpcds.warehouse_t12;
openGauss=# DROP TABLE tpcds.warehouse_t13;
openGauss=# DROP TABLE tpcds.warehouse_t14;
openGauss=# DROP TABLE tpcds.warehouse_t15;
openGauss=# DROP TABLE tpcds.warehouse_t16;
openGauss=# DROP TABLE tpcds.warehouse_t17;
openGauss=# DROP TABLE tpcds.warehouse_t18;
openGauss=# DROP TABLE tpcds.warehouse_t20;
openGauss=# DROP TABLE tpcds.warehouse_t21;
openGauss=# DROP TABLE tpcds.warehouse_t22;
openGauss=# DROP TABLE joe.warehouse_t23;
openGauss=# DROP TABLE tpcds.warehouse_t24;
openGauss=# DROP TABLE tpcds.warehouse_t25;
openGauss=# DROP TABLE tpcds.warehouse_t26;
openGauss=# DROP TABLE tpcds.warehouse_t27;
openGauss=# DROP TABLE tpcds.warehouse_t28;
openGauss=# DROP TABLE creditcard_info;
openGauss=# DROP TABLESPACE DS_TABLESPACE1;
openGauss=# 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 does not have the primary/standby mechanism. If the database is restarted due to a system fault or abnormal breakpoint, data in the unlogged table will be deleted, which may cause data loss. Therefore, the unlogged table cannot be used to store basic data.
- **TEMPORARY | TEMP**
 - A temporary table is automatically dropped at the end of the current 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 DN. That is, each DN has complete table data. The syntax is **distribute by replication**.

7.13.66 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.
- For details about how to prepare for creating tablespaces, see the description of parameters below.
- 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 an ordinary tablespace is as follows:

```
WITH ( {filesystem= { 'general'| "general" | general} |  
random_page_cost = { 'value ' | value } |  
seq_page_cost = { '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 that complies with the [Identifier Naming Conventions](#).

- **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 the name 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: CNs and DNs *data directory/pg_location/relative path* 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 a 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.
openGauss=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'test_tablespace/test_tablespace_1';

-- Create user joe.
openGauss=# CREATE ROLE joe IDENTIFIED BY '*****';

-- Create user jay.
openGauss=# CREATE ROLE jay IDENTIFIED BY '*****';

-- Create a tablespace and set its owner to user joe.
openGauss=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'test_tablespace/
test_tablespace_2';

-- Rename the ds_location1 tablespace to ds_location3.
openGauss=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

-- Change the owner of the ds_location2 tablespace.
openGauss=# ALTER TABLESPACE ds_location2 OWNER TO jay;

-- Drop tablespaces.
openGauss=# DROP TABLESPACE ds_location2;
openGauss=# DROP TABLESPACE ds_location3;

-- Drop users.
openGauss=# DROP ROLE joe;
openGauss=# 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.

7.13.67 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 [ UNLOGGED ] TABLE table_name
[ ( column_name [, ...] ) ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | { [HASH] ( column_name ) } } ]
[ TO { GROUP groupname | NODE ( nodename [, ...] ) } ]
AS query
[ WITH [ NO ] DATA ];
```

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, it is also insecure. After the database is restarted due to a conflict or abnormal shutdown, the data in the unlogged table is cleared. Contents of an unlogged table are also not replicated to standby nodes. 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 an abnormal shutdown or other unexpected operations, 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.

- **table_name**

Specifies the name of the table to be created.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **column_name**

Specifies the name of a column to be created in the new table.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **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. 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. This parameter is valid only for row-store tables.

Value range: 10–100.

- **ORIENTATION**

Value range:

COLUMN: The data will be stored in columns.

ROW (default value): The data will be stored in rows.

- **hashbucket**

Creates a hash table that uses buckets. This parameter supports only row-store tables, including row-store range tables.

Value range: **on** and **off**

Default value: **off**

NOTICE

In current version, DDL operations on hash bucket tables are affected. Therefore, you are advised not to frequently perform DDL operations on hash bucket tables.

- **TABLESPACE tablespace_name**

Specifies that the new table will be created in the **tablespace_name** tablespace. If it is not specified, the default tablespace is used.

- **DISTRIBUTE BY**

Specifies how the table is distributed or replicated between nodes.

- **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.

NOTICE

When **DISTRIBUTE BY HASH (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.

Default value: **HASH(column_name)**. Set **column_name** to the primary key (if any) of the table or the column whose first data type supports distribution keys.

column_name supports the following data types:

- Integer types: **TINYINT**, **SMALLINT**, **INT**, **BIGINT**, and **NUMERIC/DECIMAL**
- Character types: **CHAR**, **BPCCHAR**, **VARCHAR**, **VARCHAR2**, and **NVARCHAR2**
- Date/time types: **DATE**, **TIME**, **TIMETZ**, **TIMESTAMP**, **TIMESTAMPTZ**, **INTERVAL**, and **SMALLDATETIME**
- **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.  
openGauss=# CREATE SCHEMA tpcds;  
  
-- Create the tpcds.store_returns table.
```

```
openGauss=# 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.
openGauss=# INSERT INTO tpcds.store_returns(W_WAREHOUSE_SK, W_WAREHOUSE_ID, sr_item_sk,
W_WAREHOUSE_SQ_FT) VALUES (1, 'AAAAAAAABAAAAAAA', '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.
openGauss=# 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.
openGauss=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

-- Drop the table.
openGauss=# DROP TABLE tpcds.store_returns_t1 ;
openGauss=# DROP TABLE tpcds.store_returns_t2 ;
openGauss=# DROP TABLE tpcds.store_returns;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[CREATE TABLE](#) and [SELECT](#)

7.13.68 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, hash partitioning, list partitioning, and value partitioning. Currently, row-store tables support only range 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. Currently, range partitioning only allows the use of the range partitioning policy.

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.

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.
- When querying or updating most of the records in a partition, continuously scanning that partition instead of accessing the entire table can achieve a significant performance improvement.
- 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

- When specifying a partition for query, for example, **select * from tablename partition (partitionname)**, ensure that **partition** is correct. If it is incorrect, no error is reported during the query. In this case, the query is performed based on the table alias.
- In distributed mode, you can only use SELECT to specify partitions. If other syntax is used to specify partitions, an error is reported and no alias conversion is performed.
- 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} [, ... ] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | { [ HASH ] ( column_name ) } } ]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
PARTITION BY {
  {RANGE (partition_key) ( partition_less_than_item [, ... ] )} |
  {RANGE (partition_key) ( partition_start_end_item [, ... ] )}
} [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- **column_constraint:**

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE [ index_parameters ] |
  PRIMARY KEY [ index_parameters ]
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
  [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **table_constraint:**

```
[ 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:**
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | REOPTIONS | DISTRIBUTION | ALL }
- **index_parameters:**
[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 an error if tables have identical names. The notice prompts that the table relationship already exists.
- **partition_table_name**
Specifies the name of a partitioned table.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **column_name**
Specifies the name of a column to be created in the new table.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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.
Unlike INHERITS, the new table and 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.

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.

Unlike those of **INHERITS**, 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 signaled.

- 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 copied 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 copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- If **INCLUDING RELOPTIONS** 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 new table will copy the distribution information of the source table, including distribution type and key, and the new table cannot use 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 RELOPTIONS**, 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 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**

Determines the data storage mode of the table.

Value range:

- **COLUMN**: The data will be stored in columns.
- **ROW** (default value): The data will be stored in rows.

NOTICE

ORIENTATION cannot be modified.

– **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.

– **hashbucket**

Creates a hash table that uses buckets. This parameter supports only row-store tables, including row-store range tables.

Value range: **on** and **off**

Default value: **off**

NOTICE

In current version, DDL operations on hash bucket tables are affected. Therefore, you are advised not to frequently perform DDL operations on hash bucket tables.

• **TABLESPACE tablespace_name**

Specifies that the new table will be created in the **tablespace_name** tablespace. 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.

NOTICE

When **DISTRIBUTE BY HASH (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.

Default value: **HASH(column_name)**. Set **column_name** to the primary key (if any) of the table or the column whose first data type supports distribution keys.

column_name supports the following data types:

- INTEGER TYPES: TINYINT, SMALLINT, INT, BIGINT, NUMERIC/DECIMAL
- CHARACTER TYPES: CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2
- DATA/TIME TYPES: DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL, SMALLDATETIME

- **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(partition_key)**

Creates a range partition. **partition_key** is the name of the partition key.

(1) Assume that the VALUES LESS THAN syntax is used.

NOTICE

In this case, a maximum of four partition keys are supported.

Data types supported by the partition keys are as follows: **SMALLINT**, **INTEGER**, **BIGINT**, **DECIMAL**, **NUMERIC**, **REAL**, **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

If the START END clause is used, a range partitioning policy supports only a one-column partition key.

Data types supported by the partition key are as follows: **SMALLINT**, **INTEGER**, **BIGINT**, **DECIMAL**, **NUMERIC**, **REAL**, **DOUBLE PRECISION**, **TIMESTAMP[(p)] [WITHOUT TIME ZONE]**, **TIMESTAMP[(p)] [WITH TIME ZONE]**, and **DATE**.

- **PARTITION partition_name VALUES LESS THAN ({ partition_value | 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.
-
- **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 (32767).
 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.
-
- **{ ENABLE | DISABLE } ROW MOVEMENT**
Specifies whether to enable 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 under 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 second 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**

Specifies that the column value cannot be **NULL**. **ENABLE** can be omitted.

- **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 (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 value, 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 that 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 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.

NOTE

If **DISTRIBUTE BY REPLICATION** is not specified, the column table that contains only unique values must contain distribution keys.

- **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.

NOTE

If **DISTRIBUTE BY REPLICATION** is not specified, the column set with a primary key constraint must contain distribution keys.

- **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.

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: $wr_returned_date_sk < 2450815$, $2450815 \leq wr_returned_date_sk < 2451179$, $2451179 \leq wr_returned_date_sk < 2451544$, $2451544 \leq wr_returned_date_sk < 2451910$, $2451910 \leq wr_returned_date_sk < 2452275$, $2452275 \leq wr_returned_date_sk < 2452640$, $2452640 \leq wr_returned_date_sk < 2453005$, and $wr_returned_date_sk \geq 2453005$.

```
-- Create a temporary schema.
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;

-- Create the tpcds.web_returns table.
openGauss=# 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.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

-- Drop the P8 partition.
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;
```

```
-- Add a partition WR_RETURNED_DATE_SK with values ranging from 2453005 to 2453105.
openGauss=# 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.
openGauss=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN
(MAXVALUE);

-- Drop the P8 partition.
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION FOR (2453005);

-- Rename the P7 partition to P10.
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

-- Rename the P6 partition to P11.
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

-- Query the number of rows in the P10 partition.
openGauss=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
count
-----
0
(1 row)

-- Query the number of rows in the P1 partition.
openGauss=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
count
-----
0
(1 row)

-- Drop tables and tablespaces.
openGauss=# DROP TABLE tpcds.web_returns_p1;
openGauss=# DROP TABLE tpcds.web_returns;
openGauss=# 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.

```
openGauss=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
openGauss=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
openGauss=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
openGauss=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';

-- Create a temporary schema.
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;
openGauss=# 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.
openGauss=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING
PARTITION);

-- Change the tablespace of the P1 partition to example2.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

-- Change the tablespace of the P2 partition to example3.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

-- Split the P8 partition at 2453010.
openGauss=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
    PARTITION P9,
    PARTITION P10
);

-- Merge the P6 and P7 partitions into one.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

-- Modify the migration attribute of the partitioned table.
openGauss=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;

-- Drop tables and tablespaces.
openGauss=# DROP TABLE tpcds.web_returns_p1;
openGauss=# DROP TABLE tpcds.web_returns_p2;
openGauss=# DROP TABLE tpcds.web_returns_p3;
openGauss=# DROP SCHEMA tpcds CASCADE;
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
```

```
openGauss=# DROP TABLESPACE example3;  
openGauss=# 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 tablespaces.  
openGauss=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';  
openGauss=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';  
openGauss=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';  
openGauss=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';  
  
-- Create a temporary schema.  
openGauss=# CREATE SCHEMA tpcds;  
openGauss=# SET CURRENT_SCHEMA TO tpcds;  
  
-- Create a partitioned table with the partition key of the integer type.  
openGauss=# CREATE TABLE tpcds.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.  
openGauss=# 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_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.  
openGauss=# INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999),  
GENERATE_SERIES(0, 4999));  
openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);  
count  
-----  
1  
(1 row)  
  
openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);  
count  
-----  
500  
(1 row)  
  
-- Add partitions [5000, 5300), [5300, 5600), [5600, 5900), and [5900, 6000).  
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000)  
EVERY(300) TABLESPACE startend_tbs4;
```

```
-- Add the partition p7, specified by MAXVALUE.
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- Rename the partition p7 to p8.
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- Drop the partition p8.
openGauss=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- Rename the partition where 5950 is located to p71.
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- Split the partition [4000, 5000) where 4500 is located.
openGauss=# 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.
openGauss=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- View the partition status.
openGauss=# 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)

-- Drop tables and tablespaces.
openGauss=# DROP TABLE tpcds.startend_pt;
openGauss=# DROP SCHEMA tpcds CASCADE;
openGauss=# DROP TABLESPACE startend_tbs1;
openGauss=# DROP TABLESPACE startend_tbs2;
openGauss=# DROP TABLESPACE startend_tbs3;
openGauss=# DROP TABLESPACE startend_tbs4;
```

Helpful Links

[ALTER TABLE PARTITION](#) and [DROP TABLE](#)

7.13.69 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:
 - **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.

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 [Identifier Naming Conventions](#). 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. If the column specified by UPDATE OF contains a generated column, the trigger is also fired when the column on which the generated column depends is the target column of the UPDATE statement.
- **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 7-112 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 7-113 Special variables of PL/pgSQL trigger functions

Variable	Description
NEW	New tuple for INSERT and UPDATE operations. This variable is null for DELETE operations.
OLD	Old tuple for UPDATE and DELETE operations. This variable is null for INSERT operations.
TG_NAME	Trigger name.
TG_WHEN	Trigger timing (BEFORE , AFTER , or INSTEAD OF).
TG_LEVEL	Trigger frequency (ROW or STATEMENT).
TG_OP	Trigger event (INSERT , UPDATE , DELETE , or TRUNCATE).
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 TG_TABLE_NAME .)
TG_TABLE_NAME	Name of the table where the trigger resides.
TG_TABLE_SCHEMA	Schema of the table where the trigger resides.

Variable	Description
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 destination table.
openGauss=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
openGauss=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

-- Create a trigger function.
openGauss=# 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;

openGauss=# 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;

openGauss=# 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.
openGauss=# CREATE TRIGGER insert_trigger
BEFORE INSERT ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_insert_func();

-- Create an UPDATE trigger.
openGauss=# CREATE TRIGGER update_trigger
AFTER UPDATE ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_update_func();

-- Create a DELETE trigger.
openGauss=# 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.
openGauss=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
```



```
-- Execute the UPDATE event and check the trigger results.
openGauss=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.

-- Execute the DELETE event and check the trigger results.
openGauss=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.

-- Modify a trigger.
openGauss=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

-- Disable insert_trigger.
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

-- Disable all triggers on the current table.
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

-- Drop triggers.
openGauss=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

Helpful Links

[ALTER TRIGGER](#), [DROP TRIGGER](#), and [ALTER TABLE](#)

7.13.70 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: composite type, base type, shell type, enumerated type, and set type.

- Composite type

A composite type is specified by a list of attribute names and data types. If the data type of an attribute is collatable, the attribute's collation rule can also be specified. This is essentially the same as the row type of a table, but using CREATE TYPE avoids the need to create an actual table when all that is wanted is to define a type. A stand-alone composite type is useful as the parameter or return type of a function.

To create a composite type, you must have the USAGE permission on all of its attribute types.

- 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.

- **Set type**
It is similar to an array but has no length limit. It is mainly used in stored procedures.
- 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.)

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' [ , ... ] ] )

CREATE TYPE name AS TABLE OF data_type
```

Parameters

Composite type

- **name**
Specifies the name (optionally schema-qualified) of the type to be created.
- **attribute_name**
Specifies the name of an attribute (column) for the composite type.
- **data_type**
Specifies the name of an existing data type to become a column of the composite type. You can use **%ROWTYPE** to indirectly reference the type of a

table, or **%TYPE** to indirectly reference the type of a column in a table or composite type.

- **collation**

Specifies the name of an existing collation rule to be associated with a column of the composite type. 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.

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 oid-type parameter is the type's own OID (except for array types, which instead receive their element type's 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 written in C can be defined as 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.

- **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.)

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 as **STRICT**. Otherwise, when a **NULL** input value is read and the receive 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 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 type 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, a numeric function may 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**

Specifies whether a type is preferred within its type category. If it is, the value will be **TRUE**; otherwise, the value is **FALSE**. This parameter is optional. The default value 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 (,). Note that the delimiter is associated with the array element type, not the array type itself.

- **collatable**

(Optional) Specifies whether this type's operations can use collation information. If they can, the value will be **TRUE**. The default value is **FALSE**.

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.

 NOTE

Whenever a user-defined type is created, GaussDB automatically creates an associated array type whose name consists of the element type's name prepended with an underscore (_).

Examples

```
-- Create a composite type, create a table, insert data, and make a query.
openGauss=# CREATE TYPE compfoo AS (f1 int, f2 text);
openGauss=# CREATE TABLE t1_compfoo(a int, b compfoo);
openGauss=# CREATE TABLE t2_compfoo(a int, b compfoo);
openGauss=# INSERT INTO t1_compfoo values(1,(1,'demo'));
openGauss=# INSERT INTO t2_compfoo select * from t1_compfoo;
openGauss=# SELECT (b).f1 FROM t1_compfoo;
openGauss=# SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t2.b).f1;

-- Rename the data type.
openGauss=# ALTER TYPE compfoo RENAME TO compfoo1;

-- Change the owner of the user-defined type compfoo1 to usr1.
CREATE USER usr1 PASSWORD '*****';
openGauss=# ALTER TYPE compfoo1 OWNER TO usr1;

-- Change the schema of the user-defined type compfoo1 to usr1.
openGauss=# ALTER TYPE compfoo1 SET SCHEMA usr1;

Add a new attribute to the data type.
openGauss=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

Drop the compfoo1 type.
openGauss=# DROP TYPE usr1.compfoo1 cascade;

Drop related tables and users.
openGauss=# DROP TABLE t1_compfoo;
openGauss=# DROP TABLE t2_compfoo;
openGauss=# DROP SCHEMA usr1;
openGauss=# DROP USER usr1;

-- Create an enumerated type.
openGauss=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

-- Add a label.
openGauss=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

-- Rename the label.
openGauss=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

-- Create a set type.
openGauss=# CREATE TYPE compfoo_table AS TABLE OF compfoo;
```

Helpful Links

[ALTER TYPE](#) and [DROP TYPE](#)

7.13.71 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 a 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 *****.  
openGauss=# CREATE USER jim PASSWORD '*****';  
  
-- Alternatively, you can run the following statement:  
openGauss=# CREATE USER kim IDENTIFIED BY '*****';  
  
-- To create a user with the CREATEDB permission, add the CREATEDB keyword.  
openGauss=# CREATE USER dim CREATEDB PASSWORD '*****';  
  
-- Change the login password of user jim from ***** to *****.  
openGauss=# ALTER USER jim IDENTIFIED BY '*****' REPLACE '*****';  
  
-- Add the CREATEROLE permission to jim.  
openGauss=# ALTER USER jim CREATEROLE;  
  
-- Lock jim.  
openGauss=# ALTER USER jim ACCOUNT LOCK;  
  
-- Drop users.  
openGauss=# DROP USER kim CASCADE;
```

```
openGauss=# DROP USER jim CASCADE;  
openGauss=# DROP USER dim CASCADE;
```

Helpful Links

[ALTER USER](#), [CREATE ROLE](#), and [DROP USER](#)

7.13.72 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 with 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 that complies with the [Identifier Naming Conventions](#).
- **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 that complies with the [Identifier Naming Conventions](#).
- **view_option_name [= view_option_value]**
Specifies an optional parameter for a view.
Currently, **view_option_name** supports only the **security_barrier** parameter. Use this parameter when the view provides 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 a DDL operation that will change 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.
openGauss=# CREATE VIEW myView AS
  SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

-- Query the view.
openGauss=# SELECT * FROM myView ;

-- Drop the myView view.
openGauss=# DROP VIEW myView;
```

Helpful Links

[ALTER VIEW](#) and [DROP VIEW](#)

7.13.73 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.  
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');  
  
-- Insert multiple weak passwords into the gs_global_config system catalog.  
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');  
  
-- Clear all weak passwords in the gs_global_config system catalog.  
openGauss=# DROP WEAK PASSWORD DICTIONARY;  
  
-- View existing weak passwords.  
openGauss=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

Helpful Links

[13.14.119-DROP WEAK PASSWORD DICTIONARY](#)

7.13.74 CURSOR

Function

CURSOR defines a cursor to retrieve a small number of rows at a time out of a larger 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 ;
```

Parameter Description

- **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 **FETCH**.

Helpful Links

[FETCH](#)

7.13.75 DEALLOCATE

Description

DEALLOCATE deallocates a previously prepared statement. If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

The **PREPARE** keyword is always ignored.

Precautions

None

Syntax

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

Parameters

- **name**
Specifies the name of the prepared statement to be deallocated.
- **ALL**
Deallocates all prepared statements.

Examples

None

7.13.76 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 to the context area. With cursors, stored procedures can control alterations in context area.

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**.
- **query**
Uses the SELECT or VALUES clause to specify the rows to be returned by a 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 "FETCH."

Helpful Links

[BEGIN](#) and [FETCH](#)

7.13.77 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, and must have the SELECT permission on any table in the USING clause or whose values are read in **condition**.
- For row-store tables, DELETE can be used if the tables have primary key constraints or the execution plan can be pushed down.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] table_name [ * ] [ [ AS ] alias ]  
    [ USING using_list ]  
    [ WHERE condition | WHERE CURRENT OF cursor_name ]  
    [ 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** clause
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 target table.
Value range: an existing table name.
- **alias**
Specifies the alias of the target table.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **using_list**
Specifies a 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 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.
- **WHERE CURRENT OF cursor_name**
This parameter is reserved.
- **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. Write * to return all columns.

- **output_name**

Specifies a name to use for a returned column.

Value range: a string that complies with the [Identifier Naming Conventions](#).

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAAABAAAAAAA', '18', 'Jackson'),
(10000, 'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(15000, 'AAAAAAAADAAAAAAA', '585', 'Dogwood
Washington');

-- Create the tpcds.customer_address_bak table.
openGauss=# 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.
openGauss=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;

-- Delete all data from the tpcds.customer_address_bak table.
openGauss=# DELETE FROM tpcds.customer_address_bak;

-- Drop the tpcds.customer_address_bak table.
openGauss=# DROP TABLE tpcds.customer_address_bak;

-- Drop the tpcds.customer_address table.
openGauss=# DROP TABLE tpcds.customer_address;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Suggestions

- delete

To delete all records in a table, use the truncate syntax.

7.13.78 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. PL/pgSQL 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 language is PL/pgSQL.
- **code**
Specifies the programming language code that can be executed. The value must be a character string.

Examples

```
-- Create the webuser user.
openGauss=# CREATE USER webuser PASSWORD '*****';

-- Grant all permissions on all views in the tpcds schema to the webuser user.
openGauss=# DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT c.relname,n.nspname FROM pg_class c,pg_namespace n
              WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
    END LOOP;
END$$;

-- Drop the webuser user.
openGauss=# DROP USER webuser CASCADE;
```

7.13.79 DROP AUDIT POLICY

Description

Drops 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 that complies with the [Identifier Naming Conventions](#).

Examples

See [Examples](#) in "CREATE AUDIT POLICY."

Helpful Links

[CREATE AUDIT POLICY](#) and [ALTER AUDIT POLICY](#)

7.13.80 DROP DATABASE

Description

Drops a database.

Precautions

- Only the database owner or a user granted 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 drop 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 "CREATE DATABASE."

Helpful Links

[CREATE DATABASE](#)

Suggestions

- drop database
Do not delete databases during transactions.

7.13.81 DROP DIRECTORY

Description

Drops a specified directory.

Precautions

When **enable_access_server_directory** is set to **off**, only the initial user is allowed to delete directory objects. When **enable_access_server_directory** is set to **on**, a user with the SYSADMIN permission, the owner of the directory object, a user who is granted the DROP permission of the directory, or a user who inherits permissions of the built-in role **gs_role_directory_drop** can delete directory objects.

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.  
openGauss=# CREATE OR REPLACE DIRECTORY dir as '/tmp/';  
  
-- Drop a foreign table.  
openGauss=# DROP DIRECTORY dir;
```

Helpful Links

[CREATE DIRECTORY](#) and [ALTER DIRECTORY](#)

7.13.82 DROP FUNCTION

Description

Drops an existing 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. System administrators have this permission by default.

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 dropped.
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 drops the objects that depend on the function.
 - **RESTRICT**: refuses to drop the function if any objects depend on it. This is the default action.

Examples

See [Examples](#) in "CREATE FUNCTION."

Helpful Links

[ALTER FUNCTION](#) and [CREATE FUNCTION](#)

7.13.83 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 drop a parameter that does not exist, an error will be reported.

Helpful Links

[ALTER GLOBAL CONFIGURATION](#)

7.13.84 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 dropped.
Value range: an existing role name.

Examples

See [Examples](#) in "CREATE ROLE."

Helpful Links

[CREATE GROUP](#), [ALTER GROUP](#), and [DROP ROLE](#)

7.13.85 DROP INDEX

Description

Drops an index.

Precautions

Only the index owner, a user of a schema where the index resides, or a user who has the INDEX permission on the table where the index resides 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 dropped.
Value range: name of an existing index.
- **CASCADE | RESTRICT**
 - **CASCADE**: automatically drops the objects that depend on the index.
 - **RESTRICT**: refuses to drop the index if any objects depend on it. This is the default action.

Examples

See [Examples](#) in "CREATE INDEX."

Helpful Links

[ALTER INDEX](#) and [CREATE INDEX](#)

7.13.86 DROP LANGUAGE

Description

Drops a procedural language.

Precautions

The current version does not support this syntax.

7.13.87 DROP MASKING POLICY

Description

Drops 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 that complies with the [Identifier Naming Conventions](#).
The value is an existing policy name.

Examples

```
-- Drop a masking policy.  
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1;  
  
-- Drop a group of masking policies.  
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

Helpful Links

[ALTER MASKING POLICY](#) and [CREATE MASKING POLICY](#)

7.13.88 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 the materialized view.
 - **RESTRICT**: refuses to delete the materialized view if any objects depend on it. This is the default action.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

-- Create a materialized view named my_mv.
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Delete the materialized view named my_mv.
openGauss=# DROP MATERIALIZED VIEW my_mv;

-- Delete the table.
openGauss=# 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](#)

7.13.89 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 an 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 dropped.
Value range: an existing node name.
- **cnnodename**
Specifies the CN name. If this parameter 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 dropped.
Value range: an existing CN name.

Helpful Links

[CREATE NODE](#) and [ALTER NODE](#)

7.13.90 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 dropped.
Value range: an existing node group name.
- **DISTRIBUTE FROM src_group_name**
If the node group to be dropped originates from the node group specified by **src_group_name**, set **src_group_name** to specify the node group to which node information will be synchronized after redistribution. This statement is used only for redistribution after a scale-out. You are advised not to use it, because it may lead to data distribution errors and node group unavailability.

Helpful Links

[CREATE NODE GROUP](#)

7.13.91 DROP OWNED

Description

Drops the database objects owned by a database role.

Precautions

- The role's permissions on all objects in the current database and shared objects (databases and tablespaces) will be revoked.
- 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.

Syntax

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

- **name**
Specifies the role name.
- **CASCADE | RESTRICT**
 - **CASCADE**: automatically drops the objects that depend on the objects to be dropped.
 - **RESTRICT**: refuses to drop the objects if other objects depend on them. This is the default action.

Helpful Links

[REASSIGN OWNED](#) and [DROP ROLE](#)

7.13.92 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 dropped.
Value range: an existing stored procedure name.

Examples

See [Examples](#) in "CREATE PROCEDURE."

Helpful Links

[CREATE PROCEDURE](#)

7.13.93 DROP RESOURCE LABEL

Description

Drops 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 that complies with the [Identifier Naming Conventions](#).

Examples

```
-- Drop a resource label.  
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1;  
  
-- Drop a group of resource labels.  
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

Helpful Links

[ALTER RESOURCE LABEL](#) and [CREATE RESOURCE LABEL](#)

7.13.94 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 dropped.
Value range: an existing role name.

Examples

```
-- Create a role manager whose password is *****.
openGauss=# CREATE ROLE manager IDENTIFIED BY '*****';

-- Create a role with its validity from January 1, 2015 to January 1, 2026.
openGauss=# CREATE ROLE miriam WITH LOGIN PASSWORD '*****' VALID BEGIN '2015-01-01' VALID
UNTIL '2026-01-01';

-- Change the password of role manager to *****.
openGauss=# ALTER ROLE manager IDENTIFIED BY '*****' REPLACE '*****';

-- Change role manager to SYSADMIN.
openGauss=# ALTER ROLE manager SYSADMIN;

-- Drop role manager.
openGauss=# DROP ROLE manager;

-- Drop role miriam.
openGauss=# DROP GROUP miriam;
```

Helpful Links

[CREATE ROLE](#), [ALTER ROLE](#), and [SET ROLE](#)

7.13.95 DROP ROW LEVEL SECURITY POLICY

Description

Drops a row-level security policy from a table.

Precautions

Only the table owner or administrators can drop 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 the row-level security policy to be dropped.
- **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 data table all_data.
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));
```

```
-- Create a row-level security policy.  
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =  
CURRENT_USER);  
  
-- Drop a row-level security policy.  
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;  
  
-- Drop the all_data table.  
openGauss=# DROP TABLE all_data;
```

Helpful Links

[ALTER ROW LEVEL SECURITY POLICY](#) and [CREATE ROW LEVEL SECURITY POLICY](#)

7.13.96 DROP SCHEMA

Function

DROP SCHEMA 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. The system administrator has this permission by default.

Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **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 **CREATE SCHEMA**.

Helpful Links

[ALTER SCHEMA](#) and [CREATE SCHEMA](#)

7.13.97 DROP SEQUENCE

Description

Drops a sequence from the current database.

Precautions

Only the owner of a sequence, the owner of the schema of the sequence, or users granted with the DROP permission on the sequence can drop the sequence. By default, system administrators have the permission to drop the sequence.

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 dropped.
- **CASCADE**
Automatically drops the objects that depend on the sequence.
- **RESTRICT**
Refuses to drop the sequence if any objects depend on it. This is the default action.

Examples

```
-- Create an ascending sequence named serial, starting from 101.  
openGauss=# CREATE SEQUENCE serial START 101;  
  
-- Drop a sequence.  
openGauss=# DROP SEQUENCE serial;
```

Helpful Links

[ALTER SEQUENCE](#) and [DROP SEQUENCE](#)

7.13.98 DROP SERVER

Description

Drops 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 dropped.
- **CASCADE | RESTRICT**
 - **CASCADE**: automatically drops the objects that depend on the data server.
 - **RESTRICT**: refuses to drop the server if any objects depend on it. This is the default action.

Examples

```
-- Create my_server.  
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;  
  
-- Drop my_server.  
openGauss=# DROP SERVER my_server;
```

Helpful Links

[ALTER SERVER](#) and [CREATE SERVER](#)

7.13.99 DROP SYNONYM

Description

Drops a synonym.

Precautions

Only the owner of a synonym or a system administrator has the **DROP SYNONYM** permission.

Syntax

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

Parameters

- **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 section "CREATE SYNONYM."

Helpful Links

[ALTER SYNONYM](#) and [CREATE SYNONYM](#)

7.13.100 DROP TABLE

Description

Drops a table.

Precautions

DROP TABLE forcibly drops the specified table and the indexes depending on the table. After the table is dropped, the functions and stored procedures that need to use this table cannot be executed. Dropping a partitioned table also drops 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 drop the specified table. System administrators have the permission to drop the specified table by default.

Syntax

```
DROP TABLE [ IF EXISTS ]  
{ [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```

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**: The objects that depend on the table, such as views, triggers, and indexes, can be dropped cascadingly. Note that joined table objects cannot be dropped cascadingly.

- **RESTRICT**: refuses to drop the table if any objects depend on it. This is the default action.

Examples

See [Examples](#) in "CREATE TABLE."

Helpful Links

[ALTER TABLE](#) and [CREATE TABLE](#)

7.13.101 DROP TABLESPACE

Description

Drops a tablespace.

Precautions

- Only the tablespace owner or a user granted the DROP permission can run the **DROP TABLESPACE** command. A 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, run **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.

7.13.102 DROP TRIGGER

Description

Drops a trigger.

Precautions

Only the owner of a trigger or a system administrator has the **DROP TRIGGER** permission.

Syntax

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```

Parameters

- **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](#)

7.13.103 DROP TYPE

Description

Drops a user-defined data type.

Precautions

Only the type owner or a user granted with the DROP permission 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 dropped.
- **CASCADE**
Automatically drops the objects (such as fields, functions, and operators) that depend on the type.
RESTRICT
Refuses to drop 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](#)

7.13.104 DROP USER

Description

Drops a user and the schema with the same name as the user.

Precautions

- **CASCADE** is used to drop the objects (excluding databases) that depend on the user. It cannot drop locked objects unless the objects are unlocked or the processes locking the objects are terminated.
- In GaussDB, the **enable_kill_query** configuration parameter exists in the **postgresql.conf** file. 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 drops 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 drops the specified user.
- If the dependent objects are other databases or reside in other databases, manually drop them before dropping the user from the current database. DROP USER cannot drop objects across databases.

- Before dropping a user, you need to drop all the objects owned by the user and revoke the user's permissions on other objects. Alternatively, you can specify **CASCADE** to drop the objects owned by the user and the granted permissions.
- In a multi-tenant scenario, the service user will also be dropped when you drop a user group. If you want to use **CASCADE**, set **CASCADE** for the service user as well. If any error is reported for one user, other users cannot be dropped either.
- If the user has an error table specified when the GDS foreign table is created, the user cannot be deleted by specifying **CASCADE** 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 dropped.
Value range: an existing username.
- **CASCADE | RESTRICT**
 - **CASCADE**: automatically drops objects that depend on the user and revokes the permissions granted to the user.
 - **RESTRICT**: refuses to drop 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 **enable_kill_query** configuration parameter exists in the **postgresql.conf** file. 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 drops 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 drops the specified user.

Examples

See [Examples](#) in "CREATE USER."

Helpful Links

[ALTER USER](#) and [CREATE USER](#)

7.13.105 DROP VIEW

Description

Forcibly drops a view from the database.

Precautions

The owner of a view, owner of the schema of the view, users granted the DROP permission on the view, or users granted the DROP ANY TABLE permission can run the **DROP VIEW** command. By default, a 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**: automatically 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](#)

7.13.106 DROP WEAK PASSWORD DICTIONARY

Function

DROP WEAK PASSWORD DICTIONARY clears all weak passwords in `gs_global_config`.

Precautions

Only the initial user, system administrator, and security administrator have the permission to execute this syntax.

Syntax

```
DROP WEAK PASSWORD DICTIONARY;
```

Description

None.

Example

See [CREATE WEAK PASSWORD DICTIONARY](#).

Helpful Links

[13.14.82-CREATE WEAK PASSWORD DICTIONARY](#)

7.13.107 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 of the same data type as that specified parameter in creating and generating the prepared statement.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the reason table.
openGauss=# CREATE TABLE tpcds.reason (
  CD_DEMO_SK      INTEGER      NOT NULL,
  CD_GENDER       character(16) ,
  CD_MARITAL_STATUS character(100)
);

-- Insert data.
```



```
openGauss=# INSERT INTO tpods.reason VALUES(51, 'AAAAAAAADDDAAAAA', 'reason 51');

-- Create the reason_t1 table.
openGauss=# CREATE TABLE tpods.reason_t1 AS TABLE tpods.reason;

-- Create a prepared statement for an INSERT statement and execute the prepared statement.
openGauss=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO
tpods.reason_t1 VALUES($1,$2,$3);

openGauss=# EXECUTE insert_reason(52, 'AAAAAAAADDDAAAAA', 'reason 52');

-- Drop the reason and reason_t1 tables.
openGauss=# DROP TABLE tpods.reason;
openGauss=# DROP TABLE tpods.reason_t1;

-- Drop the schema.
openGauss=# DROP SCHEMA tpods CASCADE;
```

7.13.108 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 SYSADMIN or MONADMIN 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 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 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 aggregate function is inconsistent with that on the CN. Multiple pieces of information are returned. The array_avg function is not supported.

Syntax

```
EXECUTE DIRECT ON ( nodename [ ... ] ) query ;
EXECUTE DIRECT ON { COORDINATORS | DATANODES | ALL } query;
```

Parameters

- **nodename**
Specifies the node name.
Value range: an existing node name.
- **query**
Specifies the SQL statement to be executed.
- **COORDINATORS**
Run the query statement on all CNs.
- **DATANODES**
Run the query statement on all DN nodes.
- **ALL**
Run the query statement on all CNs and DN nodes.

Examples

```
-- Query the node distribution status of the current cluster.
openGauss=# 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 | f |
-1736975100 | 0 | 0 | 0 | 0 |
(6 rows)

-- Query records in the tpcds.customer_address table on dn_6001_6002.
openGauss=# EXECUTE DIRECT ON(dn_6001_6002) 'select count(*) from tpcds.customer_address';
count
-----
16922
(1 row)

-- Query records in the tpcds.customer_address table.
openGauss=# SELECT count(*) FROM tpcds.customer_address;
count
-----
50000
(1 row)
```

7.13.109 EXPLAIN

Description

Shows the execution plan of an SQL statement.

The execution plan shows how the tables referenced by the SQL statement will be scanned, for example, by plain sequential scan or index scan. If multiple tables are

referenced, the execution plan also shows 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 seeing whether the planner's estimates are close to reality.

Precautions

The statement is actually executed when the **ANALYZE** option 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 ] |  
  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 one in 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.

Value range:

- **TRUE** (default): displays it.
- **FALSE**: does not display them.

- **DETAIL boolean**

Specifies whether to display DN information.

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.

Value range:

- **TRUE**: displays it.
- **FALSE** (default): does not display it.

- **TIMING boolean**

Specifies whether to display the actual startup time and time spent on the output node.

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.

- **FORMAT**

Specifies the output format.

Value range: **TEXT**, **XML**, **JSON**, and **YAML**

Default value: **TEXT**

- **PERFORMANCE**

Prints all relevant information in execution.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
openGauss=# CREATE TABLE tpcds.customer_address
(
  ca_address_sk      INTEGER      NOT NULL,
  ca_address_id      CHARACTER(16) NOT NULL
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAABAAAAAAA'),(10000,
'AAAAAAAACAAAAAAA');

-- Create the tpcds.customer_address_p1 table.
openGauss=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

-- Change the value of explain_perf_mode to normal.
openGauss=# SET explain_perf_mode=normal;

-- Display an execution plan for simple queries in the table.
openGauss=# 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).
openGauss=# 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.
openGauss=# 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).
openGauss=# 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)

-- Here is an example of a query plan with cost estimates suppressed:
openGauss=# 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)

-- Here is an example of a query plan for a query using an aggregate function:
openGauss=# 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)

-- Drop the tpcds.customer_address_p1 table.
openGauss=# DROP TABLE tpcds.customer_address_p1;

-- Drop the tpcds.customer_address table.
openGauss=# DROP TABLE tpcds.customer_address;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[ANALYZE | ANALYSE](#)

7.13.110 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 and does not print information on the screen.

Syntax

```
EXPLAIN PLAN  
[ SET STATEMENT_ID = name ]  
FOR statement ;
```

Parameters

- **name**
Specifies a query tag.
Value range: a string.
- **PLAN**: saves plan information into **PLAN_TABLE**. If information is stored successfully, "EXPLAIN SUCCESS" is returned.
- **STATEMENT_ID**: tags each query. The tag information will be stored in **PLAN_TABLE**.

NOTE

If **SET STATEMENT_ID** is not specified 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 a DN.
- 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 the **EXPLAIN PLAN** statement.

 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.
openGauss=# CREATE TABLE foo1(f1 int, f2 text, f3 text[]);
openGauss=# CREATE TABLE foo2(f1 int, f2 text, f3 text[]);

-- Run EXPLAIN PLAN.
openGauss=# 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.

```
openGauss=# SELECT * FROM plan_table;
```

Step 3 Delete data from PLAN_TABLE.

```
openGauss=# DELETE FROM plan_table WHERE STATEMENT_ID = 'TPCH-Q4';
openGauss=# DROP TABLE foo1;
openGauss=# 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.

```
explain plan set statement_id = 'test remote query' for
select
current_user
from
customer;
```

Query PLAN_TABLE.

```
SELECT * FROM PLAN_TABLE;
```

statement_id	plan_id	id	operation	options	object_name	object_type	object_owner	projection
test remote query	29360133	1	NESTED LOOPS	CARTESIAN				'apple'::name
test remote query	29360133	2	DATA NODE SCAN		customer	REMOTE_QUERY		
test remote query	29360133	3	DATA NODE SCAN		customer_address	REMOTE_QUERY		

(3 rows)

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.

```
set enable_stream_recursive = off;
```

Run the **EXPLAIN PLAN** statement.

```
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.


```
SELECT * FROM PLAN_TABLE;
```

statement_id	plan_id	id	operation	options	object_name	object_type	object_owner	projection
cte can not be push down	25166071	1	SORT					rq.id, rq.name
cte can not be push down	25166071	2	CTE SCAN		rq	CTE		rq.id, rq.name
(2 rows)								

7.13.111 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*th 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*th succeeding row or the *count*th prior row if count is negative.

Value range: a possibly-signed integer.

- If *count* is positive, the *count*th 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.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAAABAAAAAAA', '18', 'Jackson'),(2,
'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(3, 'AAAAAAAADAAAAAAA', '585', 'Dogwood Washington');

-- (For the SELECT statement, traverse a table using a cursor.) Start a transaction.
openGauss=# START TRANSACTION;

-- Set up cursor1.
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

-- Fetch the first three rows in cursor1.
openGauss=# FETCH FORWARD 3 FROM cursor1;
ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
1 | AAAAAAAAAABAAAAAAA | 18 | Jackson
2 | AAAAAAACAACAAAAAAA | 362 | Washington 6th
3 | AAAAAAADAAAAAAA | 585 | Dogwood Washington
(3 rows)
```

```
-- Close the cursor and commit the transaction.
openGauss=# CLOSE cursor1;

-- End the transaction.
openGauss=# END;

-- (For the VALUES clause, traverse the clause using a cursor.) Start a transaction.
openGauss=# START TRANSACTION;

-- Set up cursor2.
openGauss=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

-- Fetch the first two rows in cursor2.
openGauss=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
-----+-----
0 |      3
1 |      2
(2 rows)

-- Close the cursor and commit the transaction.
openGauss=# CLOSE cursor2;

-- End the transaction.
openGauss=# END;

-- (WITH HOLD cursor) Start a transaction.
openGauss=# START TRANSACTION;

-- Create a cursor using WITH HOLD.
openGauss=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER
BY 1;

-- Fetch the first two rows in cursor1.
openGauss=# FETCH FORWARD 2 FROM cursor1;
ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
1 | AAAAAAAAAABAAAAAA | 18 | Jackson
2 | AAAAAAACAACAAAAAA | 362 | Washington 6th
(2 rows)

-- End the transaction.
openGauss=# END;

-- Fetch the next row in cursor1.
openGauss=# FETCH FORWARD 1 FROM cursor1;
ca_address_sk | ca_address_id | ca_street_number | ca_street_name
-----+-----+-----+-----
3 | AAAAAAADAAAAAAA | 585 | Dogwood Washington
(1 row)

-- Close the cursor.
openGauss=# CLOSE cursor1;

-- Drop the tpcds.customer_address table.
openGauss=# DROP TABLE tpcds.customer_address;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[CLOSE](#) and [MOVE](#)

7.13.112 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, VADMIN, 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 users, but the following permissions are granted to PUBLIC users: 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 users 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 system administrators can grant or revoke permissions to or from any roles or users. Roles with the CREATEROLE permission can grant or revoke permissions to or from non-SYSADMIN 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 users. When separation of duties is disabled, the initial user and system administrators 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 PACKAGE, EXECUTE ANY PACKAGE, and CREATE ANY TYPE. For details about the ANY permission scope, see [Table 7-114](#).

Precautions

- It is not allowed to grant the ANY permissions to PUBLIC users or revoke the ANY permissions from PUBLIC users.
- The ANY permissions are database permissions and are valid only for database objects that are granted 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 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 the corresponding operation permission.
- Exercise caution when granting the CREATE ANY FUNCTION or CREATE ANY PACKAGE permission to users to prevent other users from using DEFINER functions or PACKAGE for privilege escalation.

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 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 procedural language access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }  
ON LANGUAGE lang_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];
```

 NOTE

Java and Internal can be created only by users with the SYSADMIN permission.

- 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 the package permission to a role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }  
ON PACKAGE package_name [, ...]  
TO { [ GROUP ] role_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];
```

 NOTE

In the current version, the package access permission cannot be granted.

- 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 PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]  
TO [ GROUP ] role_name [, ...]  
[ WITH ADMIN OPTION ];
```

NOTE

In the current version, the package access permission cannot be granted.

Parameters

GRANT permissions are classified as follows:

- **SELECT**
Allows SELECT from any table, view, or sequence.
- **INSERT**
Allows INSERT of a new row into a table.
- **UPDATE**
Allows UPDATE of any column of a table. SELECT ... FOR UPDATE and SELECT ... FOR SHARE also require this permission on at least one column, in addition to the SELECT permission.
- **DELETE**
Allows DELETE of a row from a table.
- **TRUNCATE**
Allows TRUNCATE on a table.
- **REFERENCES**
Allows the creation of 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 drop 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 that complies with the [Identifier Naming Conventions](#).

- **arg_name**

Specifies the parameter name.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **arg_type**

Specifies the parameter type.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **loid**

Specifies the identifier of the large object that includes this page.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **tablespace_name**

Specifies the tablespace name.

- **client_master_key**

Name of the client master key.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **column_encryption_key**

Name of the column encryption key.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **directory_name**

Specifies a directory name.

Value range: a string that complies with the [Identifier Naming Conventions](#).

- **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 system 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 7-114 ANY permissions

Permission	Description
CREATE ANY TABLE	Users can create tables or views in the public and user schemas. The users must be granted the permission to create sequences to create a table that contains serial columns.
ALTER ANY TABLE	Users' ALTER 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 the index permission for the table.
DROP ANY TABLE	Users' DROP permission on tables or views in the public and user schemas.
SELECT ANY TABLE	Users' SELECT permission on tables or views in the public and user schemas, which is still subject to row-level security.
UPDATE ANY TABLE	Users' UPDATE permission on tables or views in the public and user schemas, which is still subject to row-level security.
INSERT ANY TABLE	Users' INSERT permission on tables or views in the public and user schemas.
DELETE ANY TABLE	Users' DELETE 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' EXECUTE permission on functions or stored procedures in the public and user schemas.
CREATE ANY PACKAGE	In the current version, the package access permission cannot be granted.

Permission	Description
EXECUTE ANY PACKAGE	In the current version, the package access permission cannot be granted.
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.
CREATE ANY INDEX	Users can create indexes in the public and user schemas. The users must be granted the permission to create tablespaces to create a partitioned table index in a tablespace.

NOTE

If a user is granted 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.

```
openGauss=# CREATE USER joe PASSWORD 'xxxxxxxxxx';
openGauss=# 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**.

```
openGauss=# CREATE SCHEMA tpcds;
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk    INTEGER    NOT NULL,
  r_reason_id    CHAR(16)   NOT NULL,
  r_reason_desc  VARCHAR(20)
);
openGauss=# REVOKE ALL PRIVILEGES FROM joe;
openGauss=# GRANT USAGE ON SCHEMA tpcds TO joe;
openGauss=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

Then **joe** has all permissions on the **tpcds.reason** table, including the add, delete, modify, and query permissions.

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**.

```
openGauss=# 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:

```
openGauss=# 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.

```
openGauss=# CREATE DATABASE testdb;
openGauss=# 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.

```
openGauss=# CREATE ROLE tpcds_manager PASSWORD 'xxxxxxxxxx';
openGauss=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```

Grant all permissions on the **tpcds_tbspc** tablespace to **joe**, but do not allow **joe** to grant these permissions to others.

```
openGauss=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
openGauss=# GRANT ALL ON TABLESPACE tpcds_tbspc 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.

```
openGauss=# CREATE ROLE manager PASSWORD 'xxxxxxxxxx';
openGauss=# GRANT joe TO manager WITH ADMIN OPTION;
```

2. Create the **senior_manager** user and grant **manager**'s permissions to it.

```
openGauss=# CREATE ROLE senior_manager PASSWORD 'xxxxxxxxxx';
openGauss=# GRANT manager TO senior_manager;
```

3. Revoke permissions and drop users.

```
openGauss=# REVOKE joe FROM manager;
openGauss=# REVOKE manager FROM senior_manager;
openGauss=# DROP USER manager;
```

Example: Revoking permissions and dropping roles and users

```
openGauss=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
openGauss=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
openGauss=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
openGauss=# DROP TABLESPACE tpcds_tbspc;
openGauss=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
openGauss=# DROP ROLE tpcds_manager;
openGauss=# DROP ROLE senior_manager;
openGauss=# DROP USER joe CASCADE;
openGauss=# DROP TABLE tpcds.reason;
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[REVOKE](#) and [ALTER DEFAULT PRIVILEGES](#)

7.13.113 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 insert long strings into columns of char- and varchar-typed columns in the target table (not involving foreign tables), the system will truncate the long strings at the maximum length defined in the target table.

 NOTE

If inserting multi-byte character data (such as Chinese characters) into 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.

- **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.
- **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. Inserting data into only some columns of a composite type leaves the other columns **NULL**.
 - 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 joined with the corresponding columns from left to right in the table.Value range: name of 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 set to **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 specified default value has been 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 that complies with the [Identifier Naming Conventions](#).

- **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 used 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 not supported. The INSERT or UPDATE trigger of the target table is not fired.
- 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 keys and unique index columns cannot be updated.
- The WHERE clause of UPDATE does not contain sublinks.

Examples

```
-- Create a schema.  
openGauss=# CREATE SCHEMA tpceds;
```

```
-- Create the tpcds.reason table.
openGauss=# CREATE TABLE tpcds.reason
(
    r_reason_sk integer,
    r_reason_id character(16),
    r_reason_desc character(100)
);

-- Insert a record into a table.
openGauss=# INSERT INTO tpcds.reason(r_reason_sk, r_reason_id, r_reason_desc) VALUES (0,
'AAAAAAAAAAAAAAAA', 'reason0');

-- Create the tpcds.reason_t2 table.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAABAAAAA', 'reason1');

-- Insert a record into the table, which is equivalent to the previous syntax.
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAABAAAAA', 'reason2');

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAACA', 'reason3'), (4,
'AAAAAADAAAAA', 'reason4'), (5, 'AAAAAAEAAAAA', 'reason5');

-- Insert records whose r_reason_sk in the tpcds.reason table is less than 5.
openGauss=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

-- Create a unique index for a table.
openGauss=# 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 BBBBBBBCAAAAAA.
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBCAAAAAA', 'reason5'), (6,
'AAAAAADAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBCAAAAAA';

-- Drop the tpcds.reason_t2 table.
openGauss=# DROP TABLE tpcds.reason_t2;

-- Drop the tpcds.reason table.
openGauss=# DROP TABLE tpcds.reason;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Suggestions

- VALUES

When you use 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.

7.13.114 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, because 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 the SELECT permission on the target table. All other forms of LOCK require table-level UPDATE and/or the DELETE permission.
- There is no **UNLOCK TABLE** command. 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 7-115](#).
- 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 API.

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 7-115 Lock mode conflicts

Requested Lock Mode/ 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
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** command.

Value range: an existing table name.

- **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 on which **FOR SHARE** and **FOR UPDATE** have not been performed.
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, which is the same as **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** commands 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** commands.

This is also the default lock mode for **LOCK TABLE** commands 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.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# 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.
openGauss=# 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.
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

openGauss=# START TRANSACTION;

openGauss=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6 );

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

openGauss=# COMMIT;

-- Drop the tpcds.reason_t1 table.
openGauss=# DROP TABLE tpcds.reason_t1;

-- Drop the table.
openGauss=# DROP TABLE tpcds.reason;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.115 MOVE

Description

MOVE 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 parameters of **MOVE** and **FETCH** commands are the same. For details, see [Parameters](#) in "FETCH."

NOTE

On successful completion, the **MOVE** command returns a "**MOVE count**" tag, where *count* indicates the number of rows that the **FETCH** command with the same parameters would have returned (possibly zero).

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAAABAAAAAAA', 'Xxxxxxxx'),(2,
'AAAAAAAAACAAAAAAA', ' Xxxxxxxx'),(3, 'AAAAAAAADAAAAAAA', ' Xxxxxxxx'),(4, 'AAAAAAAEEAAAAAAA',
'Not the product that was ordered'),(5, 'AAAAAAAFAAAAAAA', 'Parts missing'),(6, 'AAAAAAGAAAAAAA',
'Does not work with a product that I have'),(7, 'AAAAAAHHAAAAAA', 'Gift exchange');

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Define a cursor cursor1.
```

```
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

-- Skip the first three rows of cursor1.
openGauss=# MOVE FORWARD 3 FROM cursor1;

-- Fetch the first four rows from cursor1.
openGauss=# FETCH 4 FROM cursor1;
r_reason_sk | r_reason_id | r_reason_desc
-----+-----
4 | AAAAAAAAEAAAAAA | Not the product that was
ordered
5 | AAAAAAAFAAAAAAA | Parts missing
6 | AAAAAAAGAAAAAAA | Does not work with a product that I
have
7 | AAAAAAAHAAAAAAA | Gift
exchange
(4 rows)

-- Close the cursor.
openGauss=# CLOSE cursor1;

-- End the transaction.
openGauss=# END;

-- Drop the table.
openGauss=# DROP TABLE tpcds.reason;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[CLOSE](#) and [FETCH](#)

7.13.116 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 on the target table and the SELECT permission on the source table.
- MERGE INTO cannot be used 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 ] }  
];
```

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 a 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 that complies with the [Identifier Naming Conventions](#).
- **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 can contain only one **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 specified default value has been 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 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.

Examples

```
-- Create the target table products and source table newproducts, and insert data into them.
openGauss=# CREATE TABLE products
(
  product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

openGauss=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrnics');
openGauss=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrnics');
openGauss=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
openGauss=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

openGauss=# CREATE TABLE newproducts
(
  product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

openGauss=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrnics');
openGauss=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- Run MERGE INTO.
openGauss=# 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.
openGauss=# SELECT * FROM products ORDER BY product_id;
 product_id | product_name | category
-----+-----+-----
      1501 | vivitar 35mm | electrnics
      1502 | olympus camera | electrnics
      1600 | play gym | toys
      1601 | lamaze | toys
      1666 | harry potter | toys
      1700 | wait interface | books
(6 rows)

-- Drop the table.
openGauss=# DROP TABLE products;
openGauss=# DROP TABLE newproducts;
```

7.13.117 PREPARE

Description

Creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When **PREPARE** 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 "EXECUTE."

Helpful Links

[DEALLOCATE](#)

7.13.118 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** command, 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](#)

7.13.119 REASSIGN OWNED

Description

Changes the owner of the database object.

REASSIGN OWNED requires that the system change the owner of all the database objects owned by 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

7.13.120 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** cannot be executed in transaction blocks.

Syntax

- Rebuild an ordinary index.
`REINDEX { INDEX | [INTERNAL] TABLE | DATABASE | SYSTEM } name [ FORCE ];`
- Rebuild an index partition.
`REINDEX { INDEX | [INTERNAL] 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 in the table has been invalidated by running **alter unusable**, the index cannot be rebuilt.
- **DATABASE**
Rebuilds all indexes within the current database.
- **SYSTEM**

Rebuilds all indexes on system catalogs within the current database. Indexes on user tables are not processed.

- **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 rebuild indexes for only the current database. Therefore, **name** must be the same as the current database name.

- **FORCE**

Deprecated 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 executed in transaction blocks.
 - REINDEX does not support separate operations on TOAST tables or TOAST indexes.
-

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer table.
openGauss=# CREATE TABLE tpcds.customer
(
  c_customer_sk      INTEGER      NOT NULL,
  c_customer_id      CHAR(16)     NOT NULL
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAAABAAAAAA'),(5,
'AAAAAAAACAAAAAA'),(10, 'AAAAAAAADAAAAAA');

-- Create a row-store table tpcds.customer_t1 and create an index on the c_customer_sk column in the
table.
openGauss=# 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_ship_to_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);

openGauss=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

openGauss=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

-- Rebuild a single index.
openGauss=# REINDEX INDEX tpcds.tpcds_customer_index1;

-- Rebuild all indexes in the tpcds.customer_t1 table:
openGauss=# REINDEX TABLE tpcds.customer_t1;

-- Drop the tpcds.customer_t1 table.
openGauss=# DROP TABLE tpcds.customer_t1;

-- Drop the table.
openGauss=# DROP TABLE tpcds.customer;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Suggestions

- **INTERNAL TABLE**
This scenario is used for fault recovery. You are advised not to perform concurrent operations.
- **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.

7.13.121 REFRESH INCREMENTAL MATERIALIZED VIEW

Function

REFRESH INCREMENTAL MATERIALIZED VIEW refreshes a materialized view in incremental mode.

Precautions

- Incremental refresh supports only fast-refresh 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.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Fast refresh the fast-refresh materialized view my_imv.
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

-- Delete the fast-refresh materialized view.
openGauss=# DROP MATERIALIZED VIEW my_imv;

-- Delete the my_table table.
openGauss=# 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](#)

7.13.122 REFRESH MATERIALIZED VIEW

Description

REFRESH MATERIALIZED VIEW refreshes materialized views in full refresh mode.

Precautions

- Full refreshing can be performed on both complete- and fast-refresh materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

Syntax

```
REFRESH MATERIALIZED VIEW mv_name;
```

Parameters

- **mv_name**
Name of the materialized view to be refreshed.

Examples

```
-- Create an ordinary table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

-- Create a complete-refresh materialized view.
```



```
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Completely refresh the complete-refresh materialized view my_mv.
openGauss=# REFRESH MATERIALIZED VIEW my_mv;

-- Completely refresh the fast-refresh materialized view my_imv.
openGauss=# REFRESH MATERIALIZED VIEW my_imv;

-- Delete the fast-refresh materialized view.
openGauss=# DROP MATERIALIZED VIEW my_imv;

-- Delete the complete-refresh materialized view.
openGauss=# DROP MATERIALIZED VIEW my_mv;

-- Delete the my_table table.
openGauss=# 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](#)

7.13.123 RELEASE SAVEPOINT

Description

RELEASE SAVEPOINT deletes a savepoint previously defined in the current transaction.

Deleting 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. Deleting a savepoint when it is no longer needed allows the system to recycle some resources earlier than transaction end.

RELEASE SAVEPOINT also deletes all savepoints that were established after the named savepoint was established.

Precautions

- Releasing a savepoint name that was not previously defined will cause 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 delete.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create a table.
openGauss=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO tpcds.table1 VALUES (3);

-- Create a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
openGauss=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
openGauss=# COMMIT;

-- Query the table content, which should contain both 3 and 4.
openGauss=# SELECT * FROM tpcds.table1;

-- Drop the table.
openGauss=# DROP TABLE tpcds.table1;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[SAVEPOINT](#) and [ROLLBACK TO SAVEPOINT](#)

7.13.124 RESET

Description

RESET restores GUC parameters to their default values. The default values are parameter default values complied 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 GUC parameter.
Value range: GUC parameters. You can view them by running the **SHOW ALL** command.
 **NOTE**
Some parameters viewed by SHOW ALL cannot be set by using 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 session authorization.
- **ALL**
Specifies all GUC parameters.

Examples

```
-- Reset timezone to the default value.  
openGauss=# RESET timezone;  
  
-- Set all parameters to their default values.  
openGauss=# RESET ALL;
```

Helpful Links

[SET](#) and [SHOW](#)

7.13.125 REVOKE

Description

REVOKE 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 the permission on a package object.

```
REVOKE [ GRANT OPTION FOR ]  
{ { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }  
ON PACKAGE package_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 PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]  
FROM [ GROUP ] role_name [, ...];
```

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** users. Therefore, revoking the **SELECT** permission on an object from **PUBLIC** users does not necessarily mean that the **SELECT** permission on 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** users, the user can still use **SELECT**.

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** permission (with **WITH GRANT OPTION**) on a table and has granted it to user B, the permission that user B holds is called dependent permission. If the permission or the grant option held by user A is revoked, the dependent permission still exists. However, all dependent permissions are 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 a 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 A's 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, 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 "GRANT."

Helpful Links

[GRANT](#)

7.13.126 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 **ROLLBACK** 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.  
openGauss=# START TRANSACTION;  
  
-- Back out all changes.  
openGauss=# ROLLBACK;
```

Helpful Links

[COMMIT](#) | [END](#)

7.13.127 ROLLBACK PREPARED

Description

Cancels a transaction ready for two-phase commit.

Precautions

- The function is only available in maintenance mode (when the 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 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](#)

7.13.128 ROLLBACK TO SAVEPOINT

Description

ROLLBACK TO SAVEPOINT rolls back to a savepoint. It implicitly deletes 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 delete a savepoint but keep the effects of the commands executed after the savepoint was established.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

Parameters

savepoint_name

Name of the savepoint to which you want to roll back.

Examples

```
-- Undo the effects of the statements executed after my_savepoint was established:
openGauss=# START TRANSACTION;
openGauss=# SAVEPOINT my_savepoint;
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;
-- Cursor positions are not affected by savepoint rollback.
openGauss=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
openGauss=# SAVEPOINT foo;
openGauss=# FETCH 1 FROM foo;
?column?
-----
1
openGauss=# ROLLBACK TO SAVEPOINT foo;
openGauss=# FETCH 1 FROM foo;
?column?
-----
2
openGauss=# RELEASE SAVEPOINT my_savepoint;
openGauss=# COMMIT;
```

Helpful Links

[SAVEPOINT](#) and [RELEASE SAVEPOINT](#)

7.13.129 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 delete a savepoint but keep the effects of the commands 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.
openGauss=# CREATE TABLE table1(a int);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (1);

-- Create a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (2);

-- Roll back the savepoint.
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (3);

-- Commit the transaction.
openGauss=# 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.
openGauss=# SELECT * FROM table1;

-- Drop the table.
openGauss=# DROP TABLE table1;

-- Create a table.
openGauss=# CREATE TABLE table2(a int);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO table2 VALUES (3);

-- Create a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table2 VALUES (4);

-- Roll back the savepoint.
openGauss=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
openGauss=# COMMIT;

-- Query the table content, which should contain both 3 and 4.
openGauss=# SELECT * FROM table2;

-- Drop the table.
openGauss=# DROP TABLE table2;
```

Helpful Links

[RELEASE SAVEPOINT](#) and [ROLLBACK TO SAVEPOINT](#)

7.13.130 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 granted the SELECT permission on the table, or users granted 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.
- Using FOR UPDATE or FOR SHARE requires 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.
- Only reference aliases in **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.
- 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 [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** in **from_item** is as follows:
PARTITION { (partition_name) |
FOR (partition_value [, ...]) }

NOTE

Specifying partitions applies only to partitioned tables.

- The sorting order **nlssort_expression_clause** is as follows:

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
```

The second parameter can be **generic_m_ci**, which supports only the case-insensitive order for English characters.

- 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.

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. 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 result set of the SELECT statement so that each row in the result set is unique.

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*.

- **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.

- **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.

- 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 subquery results.
- 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 because you can always 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[, ...])**

Abbreviation of **ON left_table.a = right_table.a AND left_table.b = right_table.b** 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.

- **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 (+):

- It can appear only in the WHERE clause.
- If a table join has been specified in the FROM clause, the operator (+) cannot be used in the WHERE clause.
- The operator (+) can work only on columns of tables or views, instead of on expressions.
- 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.

- **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
```

- **UNION clause**

Computes the set union of the rows returned by multiple SELECT statements.

The UNION clause has the following constraints:

- The default result of UNION does not contain any duplicate rows unless the ALL clause is specified.
- Multiple UNION operators in the same SELECT statement are evaluated from left to right, unless otherwise 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 the subexpression if it is enclosed in parentheses.

- **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 from 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 from 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.
- 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, specify the **UTF-8**, **GB18030**, or **GBK** encoding mode during database initiation. The statements are as follows:

initdb -E UTF8 -D ../data -locale=zh_CN.UTF-8, initdb -E GB18030 -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.

- **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 requires a shared lock rather than an 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 in **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.

In distributed scenarios, FOR UPDATE/SHAR cannot be performed on multiple tables.

- **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, 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.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

-- Insert multiple records into the table.
openGauss=# 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,'AAAAAAAACAAAAAAA','N%reason 6'),(30,'AAAAAAAACAAAAAAA','W%reason 7');

-- Obtain the temp_t temporary table by a subquery and query all records in this table.
openGauss=# 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:
openGauss=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

-- Example of a LIMIT clause: Obtain a record from the table.
openGauss=# SELECT * FROM tpcds.reason LIMIT 1;

-- Query all records and sort them in alphabetic order.
openGauss=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;
```

```
-- Use table aliases to obtain data from pg_user and pg_user_status:
openGauss=# SELECT a.username,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.rolid;

-- Example of the FULL JOIN clause: Join data in pg_user and pg_user_status.
openGauss=# SELECT a.username,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.rolid;

-- Example of the GROUP BY clause: Filter data based on query conditions, and group the results.
openGauss=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpccs.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.
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpccs.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.
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpccs.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 tpccs.reason table.
openGauss=# SELECT r_reason_sk, tpccs.reason.r_reason_desc
FROM tpccs.reason
WHERE tpccs.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpccs.reason.r_reason_desc
FROM tpccs.reason
WHERE tpccs.reason.r_reason_desc LIKE 'N%';

-- Example of the NLS_SORT clause: Sort by Chinese Pinyin.
openGauss=# SELECT * FROM tpccs.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
SCHINESE_PINYIN_M');

-- sorting in case-insensitive order (optional; only English characters are supported in the case-insensitive
order.)
openGauss=# SELECT * FROM tpccs.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
generic_m_ci');

-- Create the partitioned table tpccs.reason_p.
openGauss=# CREATE TABLE tpccs.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.
openGauss=# INSERT INTO tpccs.reason_p values(3,'AAAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAAABAAAAAAA','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 tpccs.reason_p
table.
openGauss=# SELECT * FROM tpccs.reason_p PARTITION (P_05_BEFORE);
r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
```

```
3 | AAAAAAAAAAAAAAAAAA | reason 1
4 | AAAAAAAAAAAAAAAAAA | 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.
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
count | r_reason_id
-----+-----
2 | AAAAAAAAAAAAAAAAAA
5 | AAAAAAAAAAAAAAAAAA
(2 rows)

-- Example of the GROUP BY CUBE clause: Filter data based on query conditions, and group the results.
openGauss=# 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.
openGauss=# 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.
openGauss=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
c | r_reason_id
---+-----
5 | AAAAAAAAAAAAAAAAAA
(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
AAAAAAAAAAAAAAAA or AAAAAAAAADAAAAAAAA.
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING
r_reason_id IN('AAAAAAAAAAAAAAAA','AAAAAAAAADAAAAAAAA');
count | r_reason_id
-----+-----
5 | AAAAAAAAAAAAAAAAAA
(1 row)

-- Example of the INTERSECT clause: Query records whose r_reason_id is AAAAAAAAAAAAAAAA and
whose r_reason_sk is smaller than 5.
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAAAAAAAAAA' INTERSECT
SELECT * FROM tpcds.reason_p WHERE r_reason_sk<5;
r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
4 | AAAAAAAAAAAAAAAAAA | reason 3
3 | AAAAAAAAAAAAAAAAAA | reason 1
(2 rows)

-- Example of the EXCEPT clause: Query records whose r_reason_id is AAAAAAAAAAAAAAAA and whose
r_reason_sk is greater than or equal to 4.
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAAAAAAAAAA' EXCEPT SELECT *
FROM tpcds.reason_p WHERE r_reason_sk<4;
r_reason_sk | r_reason_id | r_reason_desc
-----+-----+-----
10 | AAAAAAAAAAAAAAAAAA | reason 5
10 | AAAAAAAAAAAAAAAAAA | reason 4
4 | AAAAAAAAAAAAAAAAAA | reason 3
10 | AAAAAAAAAAAAAAAAAA | reason 2
(4 rows)

-- Create the store_returns and customer tables.
openGauss=# CREATE TABLE tpcds.store_returns (sr_item_sk int, sr_customer_id varchar(50),sr_customer_sk
int);
openGauss=# CREATE TABLE tpcds.customer (c_item_sk int, c_customer_id varchar(50),c_customer_sk int);
openGauss=# INSERT INTO tpcds.store_returns VALUES(18000);
openGauss=# INSERT INTO tpcds.customer (c_customer_id) VALUES('AAAAAAAJNGEBAAA');

-- Specify the operator (+) in the WHERE clause to indicate a left join.
```

```
openGauss=#
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.
openGauss=#
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
-----+-----
          | AAAAAAAJNGEBAAA
(1 row)

-- Specify the operator (+) in the WHERE clause to indicate a left join and add a join condition.
openGauss=# 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.
openGauss=#
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.
openGauss=#
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.
openGauss=# 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.
openGauss=# DROP TABLE tpcds.reason_p, tpcds.reason;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.131 SELECT INTO

Description

SELECT INTO creates a table based on the query result and inserts data retrieved by the query into 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**
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, it is also insecure. After the database is restarted due to a conflict or abnormal shutdown, the data in the unlogged table is cleared. Contents of an unlogged table are also not replicated to standby nodes. 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 an abnormal shutdown or other unexpected operations, users should rebuild indexes with errors.
- **INTO [UNLOGGED] [TABLE] new_table**
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, it is also insecure. After the database is restarted due to a conflict or abnormal shutdown, the data in the unlogged table is cleared. Contents of an unlogged table are also not replicated to standby nodes. Any indexes created on an unlogged table are automatically unlogged as well.
new_table specifies the name of the new table.

NOTE

For details about other parameters of SELECT INTO, see [Parameters](#) in "SELECT."

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# CREATE TABLE tpcds.reason
(
    r_reason_sk    integer,
    r_reason_id    character(16),
    r_reason_desc  character(100)
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.reason values(1,'AAAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAAABAAAAAAA','reason 2'),(3,'AAAAAAAAABAAAAAAA','reason 3'),
(4,'AAAAAAAAABAAAAAAA','reason 4'),(4,'AAAAAAAAABAAAAAAA','reason 5'),
(4,'AAAAAAAACAAAAAAA','reason 6'),(5,'AAAAAAAACAAAAAAA','reason 7');

-- Add the values that are less than 5 in the r_reason_sk column of the tpcds.reason table to the new table.
openGauss=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

-- Drop the table.
openGauss=# DROP TABLE tpcds.reason_t1, tpcds.reason;

-- Drop the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

[SELECT](#)

7.13.132 SET

Description

Modifies GUC parameters.

Precautions

Most GUC parameters can be modified at runtime by using SET, but some parameters cannot be modified after a service 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-side 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. If neither **SESSION** nor **LOCAL** appears, **SESSION** is the default value.

If this command is executed in a transaction, the effect of the command disappears after the transaction is rolled back. Once the surrounding transaction is committed, the effect will persist until the end of the session, unless the parameters are reset by another SET statement.

- **LOCAL**

Specifies that the specified parameters take effect for the current transaction. After COMMIT or ROLLBACK, session-level settings will take effect again.

The effect of this command lasts only until the end of the current transaction, whether the transaction is 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 GUC 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 GUC 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) and **DOCUMENT**.

- **config_parameter**

Specifies the name of a configurable GUC parameter. You can use SHOW ALL to view the available GUC parameters.

 NOTE

Some parameters viewed by SHOW ALL cannot be set by using 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.  
openGauss=# SET search_path TO tpceds, public;  
  
-- Set the date style to the traditional POSTGRES style (date placed before month).  
openGauss=# SET datestyle TO postgres;
```

Helpful Links

[RESET](#) and [SHOW](#)

7.13.133 SET CONSTRAINTS

Description

SET CONSTRAINTS 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 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**, the new mode takes effect retroactively: any outstanding data modifications that would have been checked at the end of the transaction are instead 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.  
openGauss=# SET CONSTRAINTS ALL DEFERRED;
```

7.13.134 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.
- This statement may add permissions to or restrict permissions of a user. If the role of a session user has the **INHERITS** attribute, it automatically has all permissions of roles that SET ROLE enables the role to be. In this case, SET ROLE physically deletes all permissions directly granted to session users and permissions of its belonging roles and only leaves permissions of the specified roles. If the role of the session user has the **NOINHERITS** attribute, SET ROLE deletes permissions directly granted to the session user and obtains permissions 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 that complies with the [Identifier Naming Conventions](#).

- **password**

Specifies the password of a role. It must comply with the password convention. Encrypted passwords are not supported.

- **RESET ROLE**

Resets the current user identifier.

Examples

```
-- Create a role paul.
openGauss=# CREATE ROLE paul IDENTIFIED BY '*****';

-- Set the current user to paul.
openGauss=# SET ROLE paul PASSWORD '*****';

-- View the current session user and the current user.
openGauss=> SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
openGauss=> RESET ROLE;

-- Drop the user.
openGauss=# DROP USER paul;
```

7.13.135 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**
Username.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **password**
Specifies the password of a role. It must comply with the password convention. Encrypted passwords are not supported.
- **DEFAULT**
Resets the identifiers of the session and current users to the initially authenticated usernames.

Examples

```
-- Create a role paul.
openGauss=# CREATE ROLE paul IDENTIFIED BY '*****!';

-- Set the current user to paul.
openGauss=# SET SESSION AUTHORIZATION paul password '*****!';

-- View the current session user and the current user.
openGauss=> SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
openGauss=> RESET SESSION AUTHORIZATION;

-- Drop the user.
openGauss=# DROP USER paul;
```

Reference

[SET ROLE](#)

7.13.136 SET TRANSACTION

Description

Sets the transaction characteristics. Transaction characteristics include transaction isolation 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 ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }  
{ ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | 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_CLAUSE**
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 can be read. It is the default value.
- **READ UNCOMMITTED**: Uncommitted data is probably read. This isolation level is provided to handle CN breakdown emergencies. On this isolation level, you are advised to only read data to prevent inconsistency.
- **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.  
openGauss=# START TRANSACTION;  
openGauss=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;  
openGauss=# COMMIT;
```

7.13.137 SHOW

Description

SHOW shows the current value of a GUC 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.
openGauss=# SHOW timezone;

-- Show all parameters.
openGauss=# SHOW ALL;

-- Show all parameters whose names contain var.
openGauss=# SHOW VARIABLES LIKE var;
```

Helpful Links

[SET](#) and [RESET](#)

7.13.138 SHUTDOWN

Description

SHUTDOWN shuts down the currently connected database node.

Precautions

- Only administrators 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.  
openGauss=# SHUTDOWN;  
  
-- Shut down the current database node in fast mode.  
openGauss=# SHUTDOWN FAST;
```

7.13.139 START TRANSACTION

Description

START TRANSACTION 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 | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }  
    | { READ WRITE | READ ONLY }  
  } [, ...]  
];
```

Format 2: BEGIN

```
BEGIN [ WORK | TRANSACTION ]  
[  
  {  
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | 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 can be read. It is the default value.
- **READ UNCOMMITTED**: Uncommitted data is probably read. This isolation level is provided to handle CN breakdown emergencies. On this isolation level, you are advised to only read data to prevent inconsistency.

- **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

```
-- Start a transaction in default mode.
openGauss=# START TRANSACTION;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

-- Start a transaction in default mode.
openGauss=# BEGIN;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ WRITE:
openGauss=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# COMMIT;
```

Helpful Links

[COMMIT | END](#), [ROLLBACK](#), and [SET TRANSACTION](#)

7.13.140 TRUNCATE

Function

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, 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 ];
```

- 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 ];
```

Parameter Description

- **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. It is the default value.
- **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).
- **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.
The value specified by **PARTITION FOR** can 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.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

-- Insert multiple records into the table.
openGauss=# 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.
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Clear the tpcds.reason_t1 table.
openGauss=# TRUNCATE TABLE tpcds.reason_t1;

-- Delete the table.
openGauss=# DROP TABLE tpcds.reason_t1;

-- Create a partitioned table.
openGauss=# 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.
openGauss=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

-- Clear the p_05_before partition.
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

-- Clear the p_15 partition.
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (15);

-- Clear the partitioned table.
openGauss=# TRUNCATE TABLE tpcds.reason_p;

-- Delete the table.
openGauss=# DROP TABLE tpcds.reason_p;

-- Delete the table.
openGauss=# DROP TABLE tpcds.reason;

-- Delete the schema.
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.141 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.
- 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.

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 ]
[ 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 [, ...] ]
[ 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.

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 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.
- **alias**
Specifies the alias of the target table.
Value range: a string that complies with the [Identifier Naming Conventions](#).
- **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. For example:

```
UPDATE foo AS f SET f.col_name = 'namecol';
```


Value range: name of 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.

- **from_list**

Specifies a list of table expressions. You can use columns of other tables in the WHERE condition. It is similar to specifying a table list in a FROM clause of a SELECT statement.

NOTICE

Note that the target table cannot appear in the **from_list**, unless you intend a self-join (in which case it must appear with an alias in the **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 bool values (non-zero values are implicitly converted to **true** and 0 is implicitly converted to **false**), which may cause unexpected results.

- **output_expression**

Specifies an expression to be computed and returned by the **UPDATE** command 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.
openGauss=# CREATE TABLE student1
(
    stuno    int,
    classno  int
)
DISTRIBUTE BY hash(stuno);

-- Insert data.
openGauss=# INSERT INTO student1 VALUES(1,1);
openGauss=# INSERT INTO student1 VALUES(2,2);
openGauss=# INSERT INTO student1 VALUES(3,3);

-- View data.
openGauss=# SELECT * FROM student1;

-- Update the values of all records.
openGauss=# UPDATE student1 SET classno = classno*2;

-- View data.
openGauss=# SELECT * FROM student1;

-- Drop the table.
openGauss=# DROP TABLE student1;
```


7.13.142 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. If a table is specified, VACUUM processes only the specified table.
- To perform VACUUM operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. By default, system administrators have this permission. 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 a system administrator). 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. Deleting all indexes, running **VACUUM FULL**, and rebuilding indexes is usually a faster choice. 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 significant increase in I/O traffic, which may affect the performance of 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 clear deleted tuples after the delay specified by **vacuum_defer_cleanup_age**.

- When **VACUUM ANALYZE** is used, a **VACUUM** operation and then an **ANALYZE** operation are performed 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 tables so 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 recycled. After executing **DELETE**, execute 1000 non-SELECT transactions, or wait for 1s and then execute one transaction. Then, run **VACUUM FULL** to the space.
- 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

- Recycle space and update statistics information, without requirements for keyword orders.

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYZE } } [,...] ) ]  
[ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

- Recycle space, without updating statistics information.

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name ] [ PARTITION  
( partition_name ) ];
```

- Recycle space and update statistics information, and require keywords in order.

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYZE } [ VERBOSE ]  
[ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

Parameters

- **FULL**

Selects "FULL" vacuum, which can recycle 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 | ANALYZE**

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 the column to be analyzed Defaults are 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. Defaults are all partitions.

Examples

```
-- Create a schema.
openGauss=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2');

-- Create an index in the tpcds.reason table.
openGauss=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

-- Vacuum the tpcds.reason table that has indexes.
openGauss=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

-- Drop the index.
openGauss=# DROP INDEX tpcds.ds_reason_index1 CASCADE;
openGauss=# DROP TABLE tpcds.reason;
openGauss=# 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.
- Before performing the VACUUM FULL operation, you are advised to delete all indexes in related tables, run **VACUUM FULL**, and then rebuild the index.

7.13.143 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, a large number of result rows can be processed.
- 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."

7.14 Appendix

7.14.1 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 CREATE [CONSTRAINT] TRIGGER command for triggers.
	pg_get_triggerdef(trigger_oid, pretty_bool)	Gets CREATE [CONSTRAINT] TRIGGER command for triggers.

8 Best Practices

8.1 Best Practices of Table Design

8.1.1 Selecting a Storage Model

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

Selecting a model for table storage is the first step of table definition. Select a proper storage model for your service based on the following table.

Storage Model	Application Scenario
Row store	Point queries (simple index-based queries that only return a few records). Scenarios requiring frequent addition, deletion, and modification operations

8.1.2 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 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.

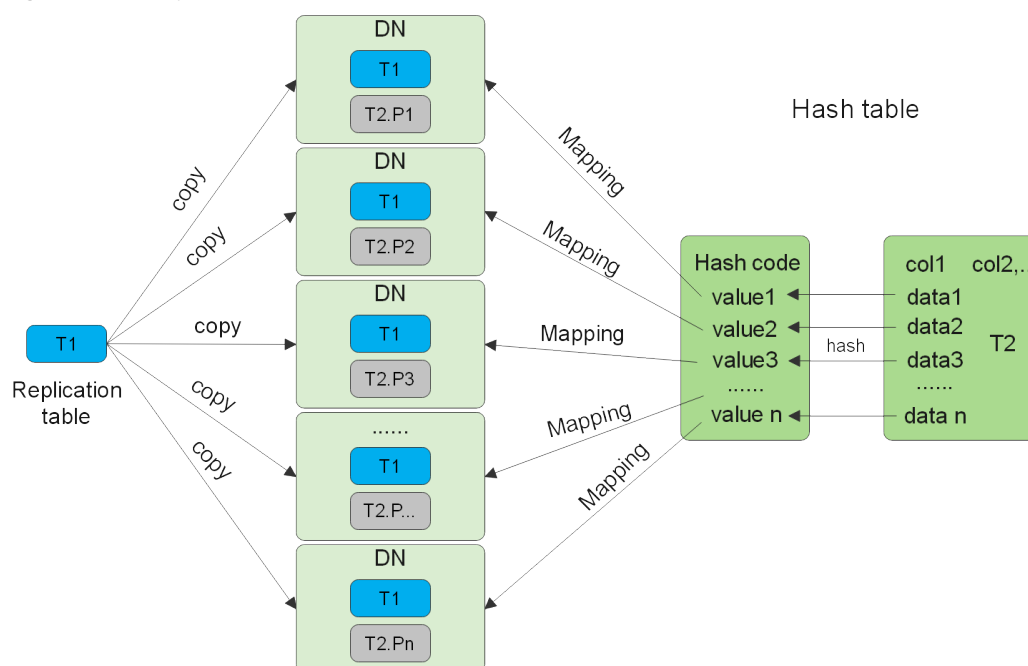
In hash mode, hash values are generated for one or more columns. You can obtain the storage location of a tuple based on the mapping between DNs and the hash values. In a hash table, I/O resources on each node can be used during data read/write, which improves the read/write speed of a table. Generally, 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 8-1](#), T1 is a replication table and T2 is a hash table.

Figure 8-1 Replication tables and hash tables



8.1.3 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. **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;
```

For 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.

8.1.4 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.

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.

8.1.5 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.

8.1.6 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 [•DISTRIBUTE BY](#). 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.
The NOT IN statement needs to be implemented using NESTLOOP ANTI JOIN, and the NOT EXISTS statement can be implemented using HASH ANTI JOIN. 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:

```
openGauss=# 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:

```
openGauss=# 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.
- 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.

```
SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
```
- 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.pclortype = '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.

This means that 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](#). However, stored procedures have no return values but the PL/SQL functions have.

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.

A GaussDB database 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.

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, 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 Arrays and Records

10.4.1 Arrays

Use of Array Types

Before the use of arrays, an array type needs to be defined:

Define an array type immediately after the **AS** keyword in a stored procedure. Run the following statement:

```
TYPE array_type IS VARRAY(size) OF data_type;
```

Related parameters are as follows:

- **array_type**: indicates the name of the array type to be defined.
- **VARRAY**: indicates the array type to be defined.
- **size**: indicates the maximum number of members in the array type to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array type to be created.

NOTE

- In GaussDB, an array automatically increases. If an access violation occurs, a **NULL** value will be returned, and no error message will be reported.
- The scope of an array type defined in a stored procedure takes effect only in this storage process.
- It is recommended that you use one of the preceding methods to define an array type. If both methods are used to define the same array type, GaussDB prefers the array type defined in a stored procedure to declare array variables.
- When **data_type** is set to a type that can define the length and precision, such as varchar and numeric, if an array of varray type is created in a package and called outside the package, the restrictions for the length or precision become invalid.
- The constructors of the array type can be used only in O-compatible mode.

GaussDB supports the access of content in an array by using parentheses, and the **extend**, **count**, **first**, **last prior**, **next**, **exists**, **trim**, and **delete** functions.

NOTE

If the stored procedure contains DML statements (**SELECT**, **UPDATE**, **INSERT**, or **DELETE**), DML statements can access array elements only using brackets. In this way, it may be separated from the function expression area.

Examples

```
-- Perform array operations in the stored procedure.
openGauss=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;-- Define the array type.
    ARRINT ARRAY_INTEGER := ARRAY_INTEGER(); -- Declare the variable of the array type.
BEGIN
    ARRINT.EXTEND(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
```



```
DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));  
ARRINT.TRIM();  
IF ARRINT.EXISTS(10) THEN  
    DBE_OUTPUT.PRINT_LINE('Exist 10th element');  
ELSE  
    DBE_OUTPUT.PRINT_LINE('Not exist 10th element');  
END IF;  
DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);  
DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));  
DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));  
ARRINT.DELETE();  
END;  
/  
  
-- Call the stored procedure.  
openGauss=# CALL array_proc();  
  
-- Delete the stored procedure.  
openGauss=# DROP PROCEDURE array_proc;
```

10.4.2 Records

Record Variables

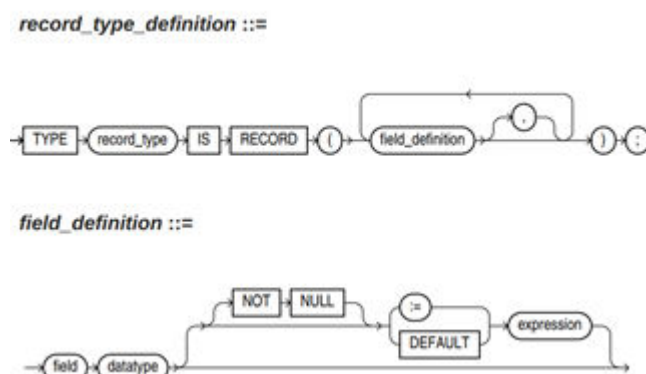
Perform the following operations to create a record variable:

Define a record type and use this type to declare a variable.

Syntax

For the syntax of the record type, see [Figure 10-1](#).

Figure 10-1 Syntax of the record type



The above syntax diagram is explained as follows:

- *record_type*: specifies the record type.
- *field*: specifies member variables of this type.
- *datatype*: specifies data types for the member variables.
- *expression*: specifies the expression for setting a default value.

 NOTE

In GaussDB:

- When assigning values to record variables, you can:
 - Declare a record type and define member variables of this type when you declare a function or stored procedure.
 - Assign the value of a record variable to another record variable.
 - Use SELECT INTO or FETCH to assign values to a record type.
 - Assign the **NULL** value to a record variable.
- The INSERT and UPDATE statements cannot use a record variable to insert or update data.
- If a member has a record type, the default value of the inner record type cannot be not transferred to the outer record type.
- Just like a variable, a record column of the compound type does not have a default value in the declaration.

Examples

The table used in the following stored procedure is defined as follows:

```
openGauss=# \d emp_rec
          Table "public.emp_rec"
  Column |          Type          | Modifiers
-----+-----+-----
 empno   | numeric(4,0)           | not null
  ename   | character varying(10)  |
   job    | character varying(9)   |
   mgr    | numeric(4,0)           |
 hiredate | timestamp(0) without time zone |
   sal    | numeric(7,2)           |
  comm    | numeric(7,2)           |
 deptno   | numeric(2,0)           |

-- Perform array operations in the stored procedure.
openGauss=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
VARCHAR2 AS $$
DECLARE

-- Declare a record type.
type rec_type is record (name varchar2(100), epno int);
employer rec_type;

-- Use %type to declare a record type.
type rec_type1 is record (name emp_rec.ename%type, epno int not null :=10);
employer1 rec_type1;

-- Declare a record type with a default value.
type rec_type2 is record (
    name varchar2 not null := 'SCOTT',
    epno int not null :=10);
employer2 rec_type2;
CURSOR C1 IS select ename,epno from emp_rec order by 1 limit 1;

BEGIN
-- Assign a value to a member record variable.
employer.name := 'WARD';
employer.epno = 18;
raise info 'employer name: % , epno:%', employer.name, employer.epno;

-- Assign the value of a record variable to another variable.
employer1 := employer;
raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

-- Assign the NULL value to a record variable.
```

```
employer1 := NULL;
raise info 'employer1 name: % , epno: %', employer1.name, employer1.epno;

-- Obtain the default value of a record variable.
raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

-- Use a record variable in the FOR loop.
for employer in select ename,empno from emp_rec order by 1 limit 1
loop
    raise info 'employer name: % , epno: %', employer.name, employer.epno;
end loop;

-- Use a record variable in the SELECT INTO statement.
select ename,empno into employer2 from emp_rec order by 1 limit 1;
raise info 'employer name: % , epno: %', employer2.name, employer2.epno;

-- Use a record variable in a cursor.
OPEN C1;
FETCH C1 INTO employer2;
raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
CLOSE C1;
RETURN employer.name;
END;
$$
LANGUAGE plpgsql;

-- Call the stored procedure.
openGauss=# CALL regress_record('abc');

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE regress_record;
```

10.5 DECLARE Syntax

10.5.1 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 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 10-2](#).
- 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.

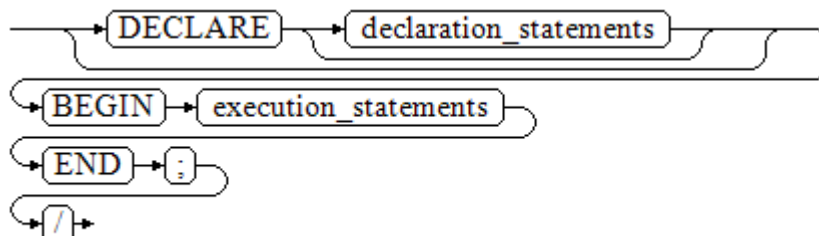
10.5.2 Anonymous Blocks

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 10-2](#) shows the syntax diagrams for an anonymous block.

Figure 10-2 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
openGauss=# BEGIN
    NULL;
END;
/

-- Display information on the console.
openGauss=# BEGIN
    dbe_output.print_line('hello world!');
END;
/

-- Display variable content on the console.
openGauss=# DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var := 'world';
    dbe_output.print_line('hello'||my_var);
END;
/
```

10.5.3 Subprograms

A subprogram stores stored procedures, functions, operators, and advanced packages.

A subprogram created in a database can be called by other programs.

10.6 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

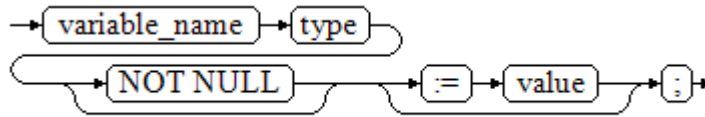
You are advised not to call the SQL statements containing passwords in the stored procedures because authorized users may view the stored procedure file in the database, causing password information leakage. If a stored procedure contains other sensitive information, permission to access this procedure must be configured, preventing information leakage.

10.6.1 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 10-3](#).

Figure 10-3 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

```
openGauss=# 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 the *my_name* variable whose data type is the same as the data type of **firstname** in **employee**, you can define the variable as follows:

```
my_name employee.firstname%TYPE
```

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.

%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
```

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.

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.

- To define a function scope, a variable must declare and create a **BEGIN-END** block in the declaration section. 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

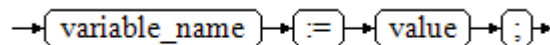
```
openGauss=# DECLARE
emp_id INTEGER :=7788; -- Define a variable and assign a value to it.
outer_var INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
DECLARE
emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
inner_var INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
dbe_output.print_line('inner emp_id ='||emp_id); -- Display the value 7799.
dbe_output.print_line('outer_var ='||outer_var); -- Reference a variable of an outer block.
END;
dbe_output.print_line('outer emp_id ='||emp_id); -- Display the value 7788.
END;
/
```

10.6.2 Assignment Statements

Variable Syntax

Figure 10-4 shows the syntax diagram for assigning a value to a variable.

Figure 10-4 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

```
openGauss=# DECLARE
emp_id INTEGER := 7788; -- Assignment
BEGIN
emp_id := 5; -- Assignment
emp_id := 5*7784;
END;
/
```

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

```
openGauss=# DROP TABLE IF EXISTS customers;
openGauss=# CREATE TABLE customers(id int,name varchar);
openGauss=# INSERT INTO customers VALUES(1,'ab');
openGauss=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; -- Assign a value.
END;
/

openGauss=# 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;
/
```

NOTICE

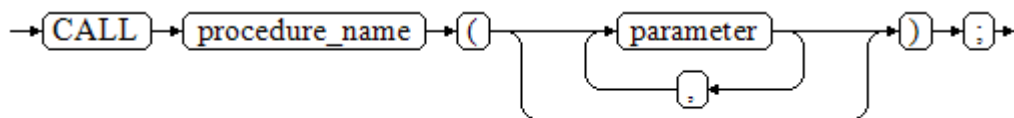
BULK COLLECT INTO can only assign values to arrays or collections in batches. Use LIMIT properly to prevent performance deterioration caused by excessive operations on data.

10.6.3 Call Statements

Syntax

Figure 10-5 shows the syntax diagram for calling a clause.

Figure 10-5 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

```
-- Create a table.
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
openGauss=# CREATE TABLE staffs
(
    section_id INTEGER,
    salary INTEGER
);
openGauss=# INSERT INTO staffs VALUES (30, 10);
openGauss=# INSERT INTO staffs VALUES (30, 20);
```



```
-- Create the stored procedure proc_staffs.
openGauss=# 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 the stored procedure proc_return.
openGauss=# 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.
openGauss=# CALL proc_return();

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE proc_staffs;
openGauss=# DROP PROCEDURE proc_return;

-- Create the function func_return.
openGauss=# CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbe_output.print_line(v_num);
RETURN; -- Return a statement.
END $$;

-- Call the function func_return.
openGauss=# CALL func_return();

-- Drop the function.
openGauss=# DROP FUNCTION func_return;

-- Drop the current database schema.
openGauss=# DROP SCHEMA hr CASCADE;
```

10.7 Dynamic Statements

10.7.1 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 10-6 shows the syntax diagram.

Figure 10-6 EXECUTE IMMEDIATE dynamic_select_clause::=

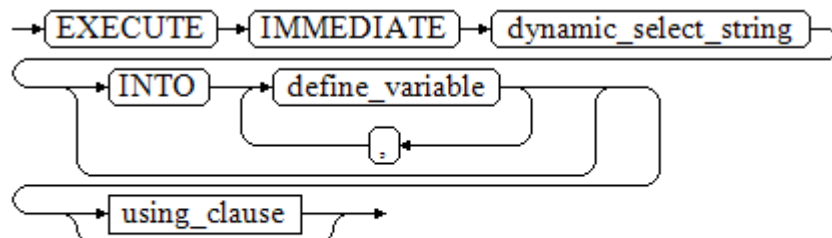
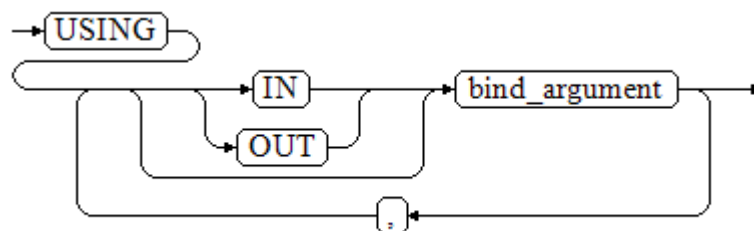


Figure 10-7 shows the syntax diagram for `using_clause`.

Figure 10-7 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 (:) followed by digits, characters, or strings, corresponding 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.

Example:

```
openGauss=# DROP SCHEMA IF EXISTS hr CASCADE;
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
openGauss=# CREATE TABLE staffs
(
    staff_id NUMBER,
    first_name VARCHAR2,
    salary NUMBER
);
openGauss=# INSERT INTO staffs VALUES (200, 'mike', 5800);
openGauss=# INSERT INTO staffs VALUES (201, 'lily', 3000);
openGauss=# INSERT INTO staffs VALUES (202, 'john', 4400);

-- Retrieve values from dynamic statements (INTO clause).
openGauss=# DECLARE
    staff_count VARCHAR2(20);
BEGIN
    EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
        INTO staff_count;
    db_output.print_line(staff_count);
END;
/

-- Pass and retrieve values (the INTO clause is used before the USING clause).
openGauss=# 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;
/

-- Call the stored procedure.
openGauss=# CALL dynamic_proc();

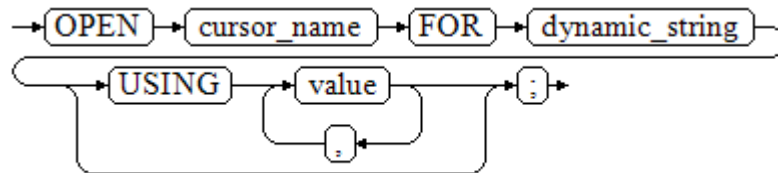
-- Drop the stored procedure.
openGauss=# DROP PROCEDURE dynamic_proc;
```

OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

Figure 10-8 shows the syntax diagram.

Figure 10-8 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:

```
openGauss=# 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.

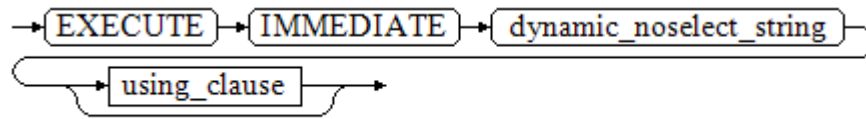
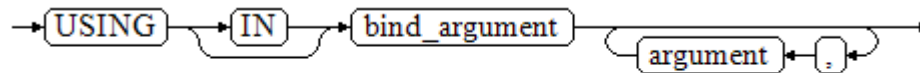
BEGIN
sqlstr := 'select first_name,phone_number,salary from hr.staffs
where section_id = :1';
OPEN my_cur FOR sqlstr USING '30'; -- Open the cursor. USING is optional.
FETCH my_cur INTO name, phone_number, salary; -- Retrieve the data.
WHILE my_cur%FOUND LOOP
dbs_output.print_line(name||' '||phone_number||' '||salary);
FETCH my_cur INTO name, phone_number, salary;
END LOOP;
CLOSE my_cur; -- Close the cursor.
END;
/

-- Drop the current database schema.
openGauss=# DROP SCHEMA hr CASCADE;
```

10.7.2 Executing Dynamic Non-Query Statements

Syntax

Figure 10-9 shows the syntax diagram.

Figure 10-9 noselect::=**Figure 10-10** shows the syntax diagram for **using_clause**.**Figure 10-10** 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*.

Examples

```

-- Create a table.
openGauss=# 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.
openGauss=# 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.
openGauss=# SELECT * FROM sections_t1;

-- Delete the table.
openGauss=# DROP TABLE sections_t1;
```

10.7.3 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 10-11 shows the syntax diagram.

Figure 10-11 call_procedure::=

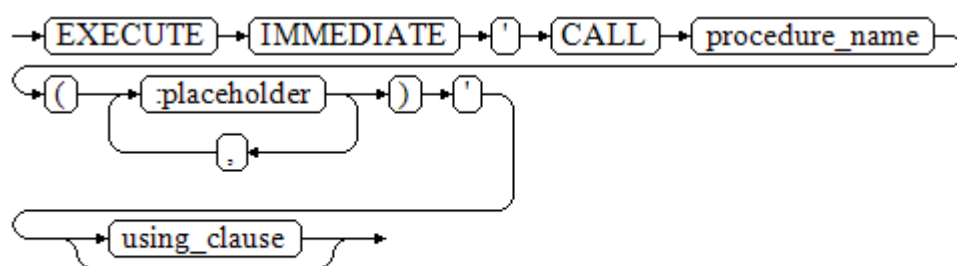
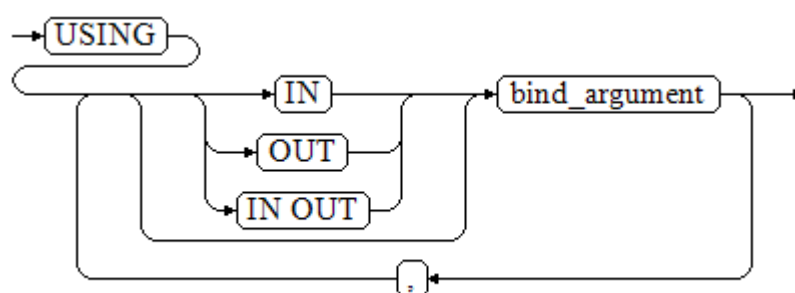


Figure 10-12 shows the syntax diagram for **using_clause**.

Figure 10-12 using_clause::=



The above syntax diagram is explained as follows:

- **CALL procedure_name**: calls the stored procedure.
- **[;placeholder1;;placeholder2,...]**: specifies the placeholder list of the stored procedure parameters. 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.

Examples

```
-- Create the stored procedure proc_add.
openGauss=# CREATE OR REPLACE PROCEDURE proc_add
(
    param1 in INTEGER,
    param2 out INTEGER,
    param3 in INTEGER
)
AS
BEGIN
    param2:= param1 + param3;
END;
/

openGauss=# 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;
/

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE proc_add;
```

10.7.4 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 10-13 shows the syntax diagram.

Figure 10-13 call_anonymous_block::=

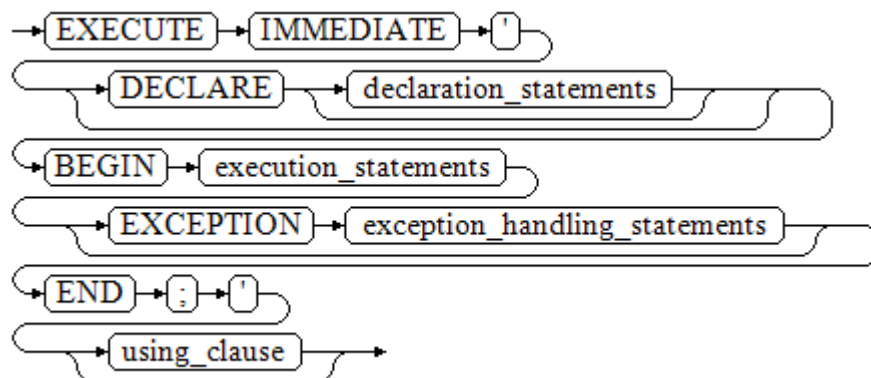
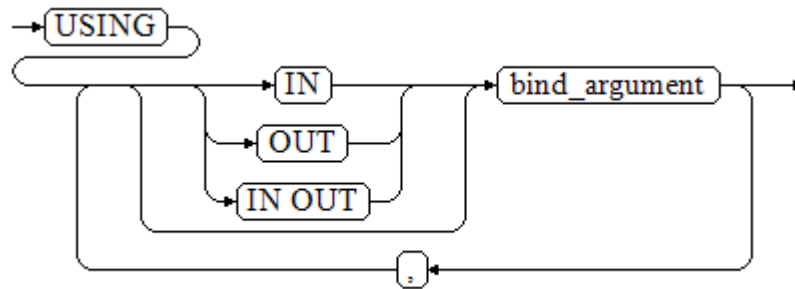


Figure 10-14 shows the syntax diagram for **using_clause**.

Figure 10-14 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 numbers of the placeholders and parameters are the same. 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.
- Variables declared in an anonymous block and binding parameters cannot be used in the same statement at the same time.
- Only SQL statements in anonymous blocks can be called to bind parameters. Parameters cannot be bound in other scenarios. For example, a stored procedure is called in an anonymous block, expressions and cursors are used in an anonymous block, and dynamic statements are called in an anonymous block.

Examples

```
-- Create a table.
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
-- Create the stored procedure dynamic_proc.
openGauss=# 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.
openGauss=# CALL dynamic_proc();

-- Drop the stored procedure.
openGauss=# DROP PROCEDURE dynamic_proc;

-- Drop the current database schema.
openGauss=# SET CURRENT_SCHEMA = public;
openGauss=# DROP SCHEMA hr CASCADE;
```

10.8 Control Statements

10.8.1 RETURN Statements

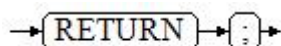
In GaussDB, data can be returned in either of the following ways: **RETURN**, **RETURN NEXT**, or **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

10.8.1.1 RETURN

Syntax

[Figure 10-15](#) shows the syntax diagram for a return statement.

Figure 10-15 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.

10.8.1.2 RETURN NEXT and RETURN QUERY

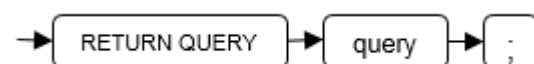
Syntax

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

```
openGauss=# CREATE TABLE t1(a int);
openGauss=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
openGauss=# 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;
openGauss=# call fun_for_return_next();
a
---
1
10
(2 rows)

-- RETURN QUERY
openGauss=# 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;
openGauss=# call fun_for_return_next();
a
---
1
10
(2 rows)

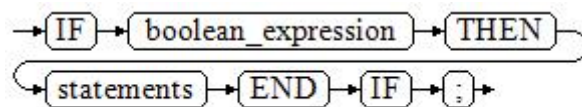
openGauss=# DROP PROCEDURE IF EXISTS fun_for_return_next();
DROP PROCEDURE
openGauss=# DROP FUNCTION IF EXISTS fun_for_return_query();
DROP FUNCTION
openGauss=# DROP TABLE t1;
DROP TABLE
```

10.8.2 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

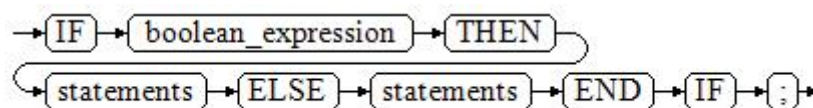
Figure 10-16 IF_THEN::=

IF_THEN is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

Example:

```
openGauss=#
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
```

- IF_THEN_ELSE

Figure 10-17 IF_THEN_ELSE::=

IF-THEN-ELSE statements add **ELSE** branches and can be executed if the condition is false.

Example:

```
openGauss=#
DECLARE
  v_user_id integer default 1;
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 NOT 0
```

- IF_THEN_ELSE IF

IF statements can be nested in the following way:

```
openGauss=#
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
```

```

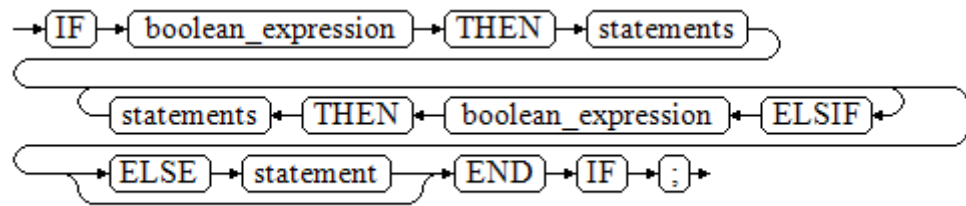
        raise info 'v_user_id > 0';
    END IF;
END IF;
END;
/
INFO: v_user_id > 0

```

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 10-18 IF_THEN_ELSIF_ELSE::=



Example:

```

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

```

- IF_THEN_ELSEIF_ELSE

ELSEIF is an alias of **ELSIF**.

Example:

```

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;
/

CALL proc_control_structure(3);

-- Drop the stored procedure.
DROP PROCEDURE proc_control_structure;

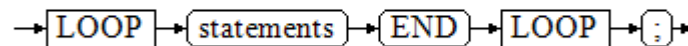
```

10.8.3 Loop Statements

Simple LOOP Statements

Syntax diagram

Figure 10-19 loop::=



Example

```

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;
/

CALL proc_loop(10,5);
  
```

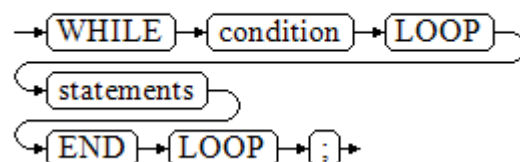
NOTICE

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

WHILE_LOOP Statements

Syntax diagram

Figure 10-20 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

```
CREATE TABLE integertable(c1 integer) DISTRIBUTE BY hash(c1);
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;
/

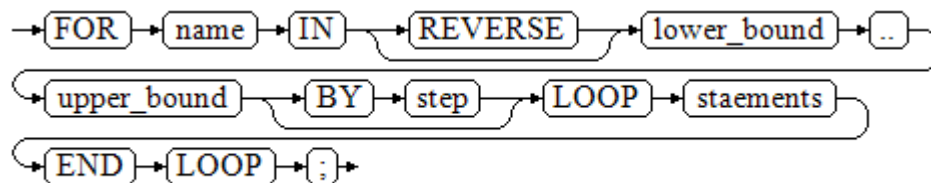
-- Call a function.
CALL proc_while_loop(10);

-- Drop the stored procedure and table.
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP (*Integer variable*) Statement

Syntax diagram

Figure 10-21 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.
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;
/

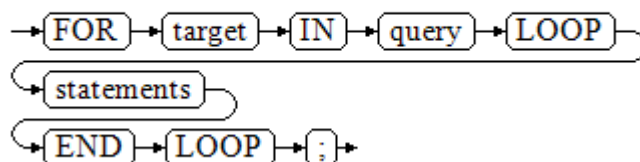
-- Call a function.
CALL proc_for_loop();

-- Drop the stored procedure.
DROP PROCEDURE proc_for_loop;
```

FOR_LOOP Query Statements

Syntax diagram

Figure 10-22 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.

Example

```

-- Display the query result from the loop.
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;
/

-- Call a function.
CALL proc_for_loop_query();

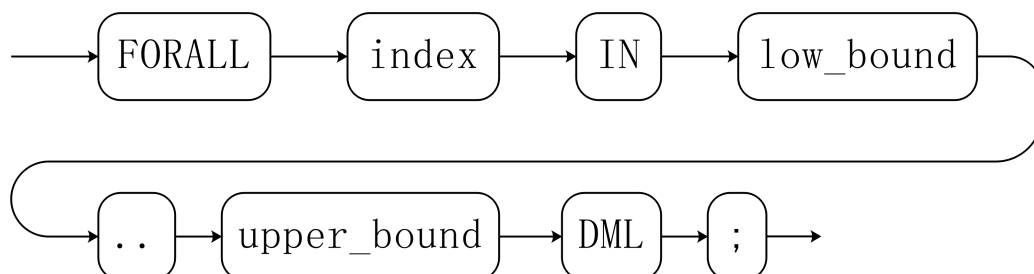
-- Drop the stored procedure.
DROP PROCEDURE proc_for_loop_query;

```

FORALL Batch Query Statements

Syntax diagram

Figure 10-23 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

```
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);

INSERT INTO TEST_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', to_date('21-06-1999',
'dd-mm-yyyy'), 'SH_CLERK' );

CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
  FORALL i IN 100..120
    update TEST_t1 set title = title + 100*i;
END;
/

-- Call a function.
CALL proc_forall();

-- Query the calling result of the stored procedure.
SELECT * FROM TEST_t1 WHERE title BETWEEN 100 AND 120;

-- Drop the stored procedure and table.
DROP PROCEDURE proc_forall;
DROP TABLE TEST_t1;
```

10.8.4 Branch Statements

Syntax

Figure 10-24 shows the syntax diagram for a return statement.

Figure 10-24 case_when::=

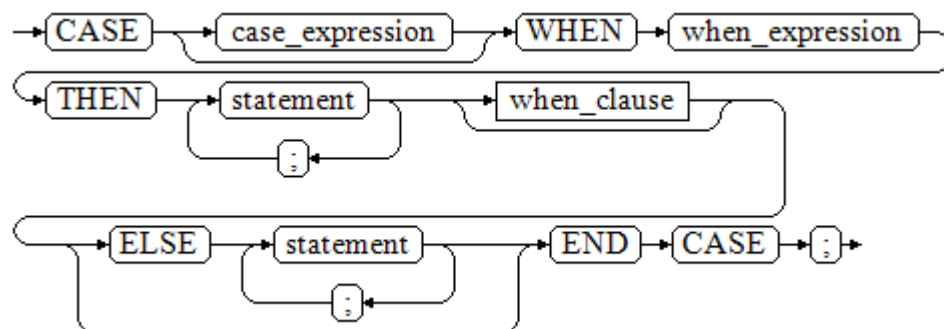
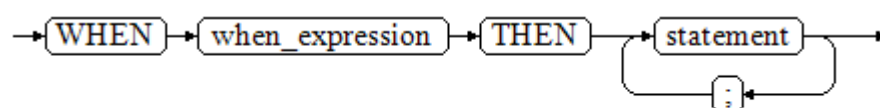


Figure 10-25 shows the syntax diagram for when_clause.

Figure 10-25 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

```
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;
/

CALL proc_case_branch(3,0);

-- Drop the stored procedure.
DROP PROCEDURE proc_case_branch;
```

10.8.5 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;
/
```

10.8.6 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:

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) DISTRIBUTE BY hash(id);

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

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;

call fun_exp();
NOTICE: caught division_by_zero
```

```
fun_exp
-----
      1
(1 row)

select * from mytab;
id | firstname | lastname
-----+-----+-----
   | Tom       | Jones
(1 row)

DROP FUNCTION fun_exp();
DROP TABLE mytab;
```

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:

```
CREATE TABLE db (a INT, b TEXT);

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;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

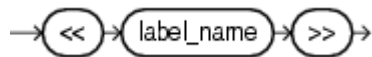
--Drop the function and table.
DROP FUNCTION merge_db;
DROP TABLE db ;
```

10.8.7 GOTO Statements

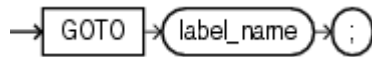
A GOTO statement unconditionally transfers the control from the current statement to a labeled statement. The GOTO statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the EXCEPTION statement to handle issues in special scenarios. To run a GOTO statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=



Examples

```
openGauss=# 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;
/

call GOTO_test();
```

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;  
/
```

10.9 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.
-

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.

```
CREATE TABLE EXAMPLE1(COL1 INT);
```

```
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.

```
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.

```
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.

```
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.
- Note: You are advised not to use the **enable_force_vector_engine** GUC parameter.

```
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;

CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
    SET enable_force_vector_engine = on;
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/

call GUC_ROLLBACK();
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
SET enable_force_vector_engine = off;
```

 NOTE

Savepoints can be used in PL/SQL stored procedures to roll back partial transaction modifications.

```
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.

```
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;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE2();
SELECT * FROM EXAMPLE1;
COMMIT;
```

 NOTE

You can use a savepoint defined outside the stored procedure to roll back to a savepoint in a PL/SQL stored procedure.

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
```



```
SAVEPOINT s1;  
INSERT INTO EXAMPLE1 VALUES(2);  
END;  
/  
  
BEGIN;  
INSERT INTO EXAMPLE1 VALUES(3);  
CALL STP_SAVEPOINT_EXAMPLE3();  
ROLLBACK TO SAVEPOINT s1; -- Roll back the insertion of record 2 to the stored procedure.  
SELECT * FROM EXAMPLE1;  
COMMIT;
```

 NOTE

The COMMIT and ROLLBACK statements can be called in a function.

```
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;  
/
```

Constraints

CAUTION

- The unapplicable contexts are as follows:
 1. COMMIT, ROLLBACK, and SAVEPOINT cannot be called in stored procedures other than PL/SQL, such as 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.
- 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.

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.

```
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;
/

CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();

DELETE FROM EXAMPLE1;
```

** NOTE**

Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT or ROLLBACK or another stored procedure that contains COMMIT or ROLLBACK.

```
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;
/
```

** NOTE**

Variables declared or imported in stored procedures cannot be committed or rolled back.

```
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.

```
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;  
/
```

** NOTE**

COMMIT/ROLLBACK cannot be called in a stored procedure whose header contains GUC parameters.

```
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;  
/
```

** NOTE**

A stored procedure object whose cursor is open cannot contain COMMIT/ROLLBACK.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT INT)  
AS  
BEGIN  
    INTOUT := INTIN + 1;  
    COMMIT;  
END;  
/
```

```
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;  
/
```

** NOTE**

COMMIT or ROLLBACK cannot be called in expressions or CURSOR and EXECUTE statements.

```
CREATE OR REPLACE PROCEDURE exec_func1()  
AS  
BEGIN  
    CREATE TABLE TEST_exec(A INT);  
    COMMIT;  
END;  
/  
CREATE OR REPLACE PROCEDURE exec_func2()  
AS  
BEGIN  
    EXECUTE exec_func1();
```

```
COMMIT;  
END;  
/
```

NOTE

Return values and expression calculation of stored procedures are not supported.

```
CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)  
AS  
BEGIN  
    RET_NUM := 1+1;  
    COMMIT;  
END;  
/  
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)  
AS  
SUM_NUM INT;  
BEGIN  
SUM_NUM := ADD_NUM + exec_func3();  
COMMIT;  
END;  
/
```

NOTE

Savepoints defined outside a stored procedure cannot be released in the stored procedure.

```
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;  
/  
  
BEGIN;  
INSERT INTO EXAMPLE1 VALUES(1);  
SAVEPOINT s1;  
CALL STP_SAVEPOINT_EXAMPLE3();  
COMMIT;
```

10.10 Other Statements

10.10.1 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](#).

10.10.2 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](#).

10.11 Cursors

10.11.1 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.

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

10.11.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when query results are multiple records.

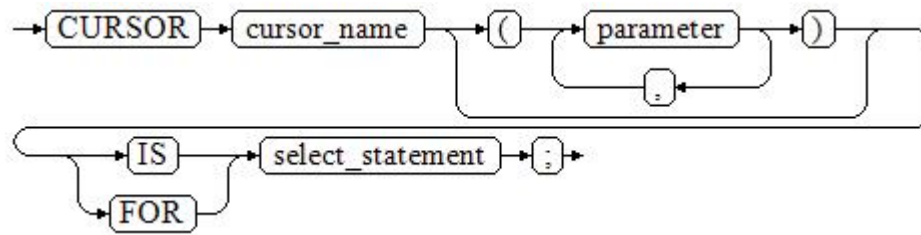
Procedure

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 10-26](#) shows the syntax diagram for defining a static cursor.

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.

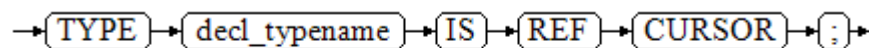
NOTE

- 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.

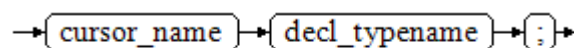
Figure 10-27 and **Figure 10-28** show the syntax diagrams for defining a dynamic cursor.

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.

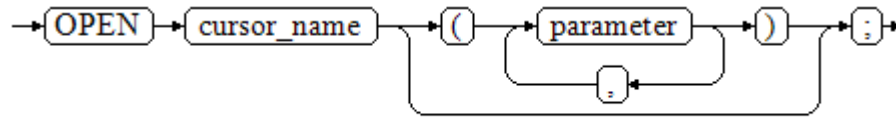
Figure 10-28 dynamic_cursor_define::=



- 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.

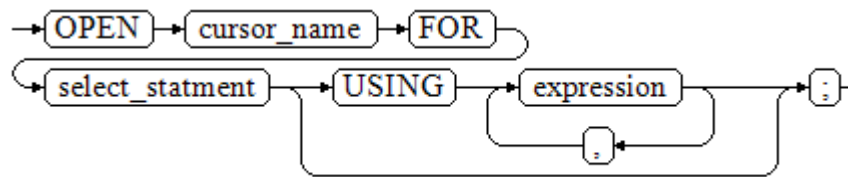
Figure 10-29 open_static_cursor::=



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::=

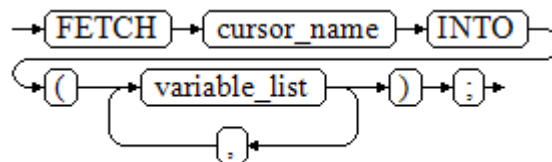


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

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::=



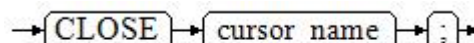
Step 4 Process the record.

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 anymore. 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 works opposite to the **%FOUND** attribute.
- **%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.

```
DROP SCHEMA IF EXISTS hr CASCADE;
CREATE SCHEMA hr;
SET current_schema = hr;
DROP TABLE IF EXISTS sections;
DROP TABLE IF EXISTS staffs;
DROP TABLE IF EXISTS department;
-- Create a department table.
CREATE TABLE sections(
    section_name varchar(100),
    place_id    int,
    section_id  int
);
INSERT INTO sections VALUES ('hr',1,1);

-- Create an employee table.
CREATE TABLE staffs(
    staff_id number(6),
    salary number(8,2),
    section_id int,
    first_name varchar(20)
);
INSERT INTO staffs VALUES (1,100,1,'Tom');

-- Create a department table.
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;

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)
-- Drop 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;
/

-- Drop the stored procedure.
DROP PROCEDURE proc_sys_ref;
```

10.11.3 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 works opposite to the **SQL%FOUND** attribute.
- **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();

-- Drop the stored procedure and the temporary table.
DROP PROCEDURE proc_cursor3;
DROP TABLE hr.staffs_t1;
DROP TABLE hr.sections_t1;
```

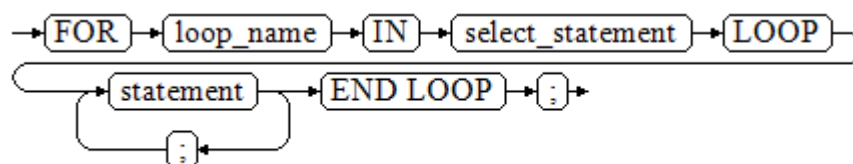
10.11.4 Cursor Loop

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 **%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;
/

-- 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.');
        END IF;
    END IF;
    CLOSE C1;-- Close the cursor.
    CLOSE C2;
END;
/

-- Delete the temporary table.
DROP TABLE integerTable1;
DROP TABLE integerTable2;

DROP SCHEMA hr CASCADE;
```

10.12 Advanced Packages

Advanced packages have two sets of APIs. The first set is basic APIs, and the second set is secondary encapsulation APIs that are used improve usability. The second set is recommended.

10.12.1 Basic APIs

10.12.1.1 PKG_SERVICE

[Table 10-3](#) lists all APIs supported by the **PKG_SERVICE** package.

Table 10-3 PKG_SERVICE

API	Description
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE	Checks whether a context is registered.
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS	Deregisters all registered contexts.
PKG_SERVICE.SQL_REGISTER_CONTEXT	Registers a context.
PKG_SERVICE.SQL_UNREGISTER_CONTEXT	Deregisters a context.
PKG_SERVICE.SQL_SET_SQL	Sets an SQL statement for a context. Currently, only the SELECT statement is supported.
PKG_SERVICE.SQL_RUN	Executes the configured SQL statement on a context.
PKG_SERVICE.SQL_NEXT_ROW	Reads the next row of data in a context.
PKG_SERVICE.SQL_GET_VALUE	Reads a dynamically defined column value in a context.
PKG_SERVICE.SQL_SET_RESULT_TYPE	Dynamically defines a column of a context based on the type OID.
PKG_SERVICE.JOB_CANCEL	Removes a scheduled task by task ID.
PKG_SERVICE.JOB_FINISH	Disables or enables scheduled task execution.
PKG_SERVICE.JOB_SUBMIT	Commits a scheduled task. Job ID can be automatically generated by the system or specified manually.
PKG_SERVICE.JOB_UPDATE	Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval.
PKG_SERVICE.SUBMIT_ON_NODES	Commits a task to all nodes. The task ID is automatically generated by the system.
PKG_SERVICE.ISUBMIT_ON_NODES	Commits a job to all nodes. The job ID is specified by the user.
PKG_SERVICE.SQL_GET_ARRAY_RESULT	Obtains the array value returned in the context.
PKG_SERVICE.SQL_GET_VARIABLE_RESULT	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 specified 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 1 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.

Parameter	Description
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(  
job 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.

Example:

```
SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');  
job_submit  
-----  
101  
(1 row)  
CALL PKG_SERVICE.JOB_CANCEL(101);  
job_cancel  
-----  
(1 row)
```

- **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.

Parameter	Type	Input/Output Parameter	Empty or Not	Description
broken	Boolean	IN	No	Specifies the status flag, true for broken and false for not broken. The current job is updated based on the parameter value true or false . 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 broken is set to true , next_time is updated to '4000-1-1'. If broken is set to false and next_time is not empty, next_time is updated for the job. If next_time 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 NULL , a job ID is generated internally.

Parameter	Type	Input/Output Parameter	Empty or Not	Description
context	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 (sysdate). 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 sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left empty or set to null , 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 pkg_service.job_submit is called using select , 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.
next_time	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left empty, the system does not update the next_time parameter for the specified job. Otherwise, the system updates the next_time 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 interval_time parameter for the specified job. Otherwise, the system updates the interval_time parameter for the specified job after necessary validity check. If this parameter is set to null , the job will be executed only once, and the job status will change to 'd' afterward.

Parameter	Type	Input/Output Parameter	Empty or Not	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 content parameter for the specified job. Otherwise, the system updates the content parameter for the specified job.

Example:

```
CALL PKG_SERVICE.JOB_UPDATE(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
job_update
-----
(1 row)
CALL PKG_SERVICE.JOB_UPDATE(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
job_update
-----
(1 row)
```

- **PKG_SERVICE.SUBMIT_ON_NODES**

The stored procedure SUBMIT_ON_NODES creates a scheduled task on all CNs and DN. 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 'ALL_NODE' (indicating that the job is executed on all nodes) or 'CCN' (indicating that the job is executed on the central coordinator node).

Parameter	Type	Input/Output Parameter	Empty or Not	Description
database	text	IN	No	Database used by a cluster job. When the node type is ALL_NODE , the value can only be postgres .
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 (sysdate). 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 sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left empty or set to null , 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 <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.
result_type	Return type.

10.12.1.2 PKG_UTIL

Table 10-18 lists all APIs supported by the PKG_UTIL package.

Table 10-18 PKG_UTIL

API	Description
PKG_UTIL.LOB_GET_LENGTH	Obtains the length of a LOB.
PKG_UTIL.LOB_READ	Reads a part of a LOB.
PKG_UTIL.LOB_WRITE	Writes the source object to the target object in the specified format.
PKG_UTIL.LOB_APPEND	Appends a specified number of characters of the source LOB to the target LOB.
PKG_UTIL.LOB_COMPARE	Compares two LOBs based on the specified length.
PKG_UTIL.LOB_MATCH	Returns the position of the <i>M</i> th occurrence of a character string in a LOB.
PKG_UTIL.LOB_RESET	Resets the character in specified position of a LOB to a specified character.
PKG_UTIL.IO_PRINT	Displays character strings.
PKG_UTIL.RAW_GET_LENGTH	Obtains the length of RAW data.
PKG_UTIL.RAW_CAST_FROM_VARCHAR2	Converts VARCHAR2 data to RAW data.
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER	Converts binary integers to RAW data.
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER	Converts RAW data to binary integers.
PKG_UTIL.RANDOM_SET_SEED	Sets a random seed.
PKG_UTIL.RANDOM_GET_VALUE	Returns a random value.
PKG_UTIL.FILE_SET_DIRNAME	Sets the directory to be operated.
PKG_UTIL.FILE_OPEN	Opens a file based on the specified file name and directory.
PKG_UTIL.FILE_SET_MAX_LINE_SIZE	Sets the maximum length of a line to be written to a file.
PKG_UTIL.FILE_IS_CLOSE	Checks whether a file handle is closed.
PKG_UTIL.FILE_READ	Reads data of a specified length from an open file handle.
PKG_UTIL.FILE_READLINE	Reads a line of data from an open file handle.

API	Description
PKG_UTIL.FILE_WRITE	Writes the data specified in the buffer to a file.
PKG_UTIL.FILE_WRITELINE	Writes the buffer to a file and adds newline characters.
PKG_UTIL.FILE_NEWLINE	Adds a line.
PKG_UTIL.FILE_READ_RAW	Reads binary data of a specified length from an open file handle.
PKG_UTIL.FILE_WRITE_RAW	Writes binary data to a file.
PKG_UTIL.FILE_FLUSH	Writes data from a file handle to a physical file.
PKG_UTIL.FILE_CLOSE	Closes an open file handle.
PKG_UTIL.FILE_REMOVE	Deletes a physical file. To do so, you must have the corresponding permission.
PKG_UTIL.FILE_RENAME	Renames files on the disk, similar to mv in Unix.
PKG_UTIL.FILE_SIZE	Returns the size of a file.
PKG_UTIL.FILE_BLOCK_SIZE	Returns the number of blocks contained in a file.
PKG_UTIL.FILE_EXISTS	Checks whether a file exists.
PKG_UTIL.FILE_GETPOS	Specifies the offset of a returned file, in bytes.
PKG_UTIL.FILE_SEEK	Sets the offset for file position.
PKG_UTIL.FILE_CLOSE_ALL	Closes all file handles opened in a session.
PKG_UTIL.EXCEPTION_REPORT_ERROR	Throws an exception.

- [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 anyelement  
)  
RETURN INTEGER;
```

Table 10-19 PKG_UTIL.LOB_GET_LENGTH API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	clob / 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/ 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.
mode	int	IN	No	Specifies the type of the read operation. 0 indicates READ , 1 indicates TRIM , and 2 indicates SUBSTR .

- 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 int  
)  
RETURN BLOB;
```

```
PKG_UTIL.LOB_WRITE(  
dest_lob INOUT clob,  
src_lob IN varchar2  
len IN int,  
start_pos IN int  
)  
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	int	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	blob / clob	INOUT	No	Specifies the target BLOB/CLOB that data will be written to.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
src_lob	blob / clob	IN	No	Specifies the source BLOB/CLOB to be written.
len	int	IN	Yes	Length of the source object to be written. If the value is NULL , the entire source object is written by default.

- **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 lob1 .
start_pos2	int	IN	No	Specifies the start offset of lob2 .

- **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.
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: 0 .

- **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 10-26 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 10-27 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 10-28 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 10-29 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 1 (BIG_ENDIAN) or 2 (LITTLE_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 10-30 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.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
endianess	INTEGER	IN	No	The value is an integer in lexicographic order. Currently, the value can be 1 (BIG_ENDIAN) or 2 (LITTLE_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 10-31 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 10-32 PKG_UTIL.FILE_SET_DIRNAME API parameters

Parameter	Description
dirname	Directory of a file. It is a string, indicating an object name. NOTE File directories need to be added to the system catalog PG_DIRECTORY . If the input path does not match the path in PG_DIRECTORY , an error indicating that the path does not exist is reported. Functions that involve location 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 10-33 PKG_UTIL.FILE_OPEN API parameters

Parameter	Description
file_name	File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the OPEN function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).
open_mode	File opening mode, including r (read), w (write), and a (append). NOTE 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 10-34 PKG_UTIL.FILE_SET_MAX_LINE_SIZE API parameters

Parameter	Description
max_line_size	Maximum number of characters in each line, including newline characters. The minimum value is 1 and the maximum is 32767 . If this parameter is not specified, the default value 1024 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 10-35 PKG_UTIL.FILE_IS_CLOSE API parameters

Parameter	Description
file	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 10-36 PKG_UTIL.FILE_READ API parameters

Parameter	Description
file	File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.
buffer	Buffer used to receive data.
len	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 10-37 PKG_UTIL.FILE_READLINE API parameters

Parameter	Description
file	File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.
buffer	Buffer used to receive data.
len	Number of bytes read from a file. The default value is NULL . If the default value NULL is used, max_line_size 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 10-38 PKG_UTIL.FILE_WRITE API parameters

Parameter	Description
file	Opened file handle.
buffer	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 PUT operations cannot exceed 32767 bytes. NOTE 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 10-39 PKG_UTIL.FILE_NEWLINE API parameters

Parameter	Description
file	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 10-40 PKG_UTIL.FILE_WRITELINE API parameters

Parameter	Description
file	Opened file handle.
buffer	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 10-41 PKG_UTIL.FILE_READ_RAW API parameters

Parameter	Description
file	Opened file handle.
length	Length of the data to be read. The default value is NULL . 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 10-42 PKG_UTIL.FILE_WRITE_RAW API parameters

Parameter	Description
file	Opened file handle.
r	Data to be written to the file. NOTE 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 10-43 PKG_UTIL.FILE_FLUSH API parameters

Parameter	Description
file	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 10-44 PKG_UTIL.FILE_CLOSE API parameters

Parameter	Description
file	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 10-45 PKG_UTIL.FILE_REMOVE API parameters

Parameter	Description
file_name	Name of the file to be deleted

- **PKG_UTIL.FILE_RENAME**

The function renames files 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 10-46 PKG_UTIL.FILE_RENAME API parameters

Parameter	Description
src_dir	Source file directory (case-sensitive) NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
src_file_name	Source file name
dest_dir	Target file directory (case-sensitive) NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
dest_file_name	Target file name
overwrite	The default value is false . 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 10-47 PKG_UTIL.FILE_SIZE API parameters

Parameter	Description
file_name	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 10-48 PKG_UTIL.FILE_BLOCK_SIZE API parameters

Parameter	Description
file_name	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 10-49 PKG_UTIL.FILE_EXISTS API parameters

Parameter	Description
file_name	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 10-50 PKG_UTIL.FILE_GETPOS API parameters

Parameter	Description
file	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 10-51 PKG_UTIL.FILE_SEEK API parameters

Parameter	Description
file	Opened file handle.
start	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 10-52 PKG_UTIL.FILE_CLOSE_ALL API parameters

Parameter	Description
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 10-53 PKG_UTIL.EXCEPTION_REPORT_ERROR API parameters

Parameter	Description
code	Error code displayed when an exception occurs.
log	Logs displayed when an exception occurs.
flag	Reserved. The default value is false .

- **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 10-54 PKG_UTIL.app_read_client_info API parameters

Parameter	Description
buffer	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 10-55 PKG_UTIL.app_set_client_info API parameters

Parameter	Description
str	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 10-56 PKG_UTIL.lob_converttoblob API parameters

Parameter	Description
dest_lob	Target LOB.
src_clob	CLOB to be converted.
amount	Conversion length.
dest_offset	Start position of the target LOB.
src_offset	Start position of the source CLOB.

- PKG_UTIL.lob_converttoclob

Converts a BLOB to a CLOB. *amount* indicates the conversion length.

The prototype of the PKG_UTIL.lob_converttoclob function is as follows:

```
PKG_UTIL.lob_converttoclob(  
dest_lob clob,  
src_blob blob,  
amount integer,  
dest_offset integer,  
src_offset integer  
)return text
```

Table 10-57 PKG_UTIL.lob_converttoclob API parameters

Parameter	Description
dest_lob	Target LOB.
src_blob	BLOB to be converted.
amount	Conversion length.

Parameter	Description
dest_offset	Start position of the target LOB.
src_offset	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 10-58 PKG_UTIL.lob_texttoraw API parameters

Parameter	Description
src_lob	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 10-59 PKG_UTIL.match_edit_distance_similarity API parameters

Parameter	Description
str1	First character string.
str2	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 10-60 PKG_UTIL.raw_cast_to_varchar2 API parameters

Parameter	Description
str	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 10-61 PKG_UTIL.session_clear_context API parameters

Parameter	Description
namespace	Namespace of an attribute.
client_identifier	Usually the value of client_identifier is the same as that of namespace . If this parameter is set to null , all namespaces are modified by default.
attribute	Attribute value to be cleared.

- **PKG_UTIL.session_search_context**

Searches for an attribute value.

The prototype of the PKG_UTIL.session_clear_context function is as follows:

```
PKG_UTIL.session_clear_context(  
namespace text,  
attribute text  
)  
RETURN INTEGER
```

Table 10-62 PKG_UTIL.session_clear_context API parameters

Parameter	Description
namespace	Namespace of an attribute.
attribute	Attribute value to be cleared.

- **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 10-63 PKG_UTIL.session_set_context API parameters

Parameter	Description
namespace	Namespace of an attribute.

Parameter	Description
attribute	Attribute to be set.
value	Attribute value.

- **PKG_UTIL.utility_get_time**

This function 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
```

10.12.2 Secondary Encapsulation APIs (Recommended)

10.12.2.1 DBE_LOB

APIs

Table 10-64 lists all APIs supported by the **DBE_LOB** package.

Table 10-64 DBE_LOB

API	Description
DBE_LOB.GET_LENGTH	Obtains and returns the specified length of a LOB.
DBE_LOB.OPEN	Opens a LOB and returns a LOB descriptor.
DBE_LOB.READ	Loads a part of LOB content to the buffer based on the specified length and initial position offset.

API	Description
DBE_LOB.WRITE	Copies content in the buffer to a LOB based on the specified length and initial position offset.
DBE_LOB.WRITE_APPEND	Copies content in the buffer to the end part of a LOB based on the specified length.
DBE_LOB.COPY	Copies content in a BLOB to another BLOB based on the specified length and initial position offset.
DBE_LOB.ERASE	Deletes content in a BLOB based on the specified length and initial position offset.
DBE_LOB.CLOSE	Closes a LOB descriptor.
DBE_LOB.MATCH	Returns the position of the <i>M</i> th occurrence of a character string in a LOB.
DBE_LOB.COMPARE	Compares two LOBs or a certain part of two LOBs.
DBE_LOB.SUBSTR	Reads the substring of a LOB and returns the number of read bytes or the number of characters.
DBE_LOB.STRIP	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 newlen parameter.
DBE_LOB.CREATE_TEMPORARY	Creates a temporary BLOB or CLOB.
DBE_LOB.FREETEMPORARY	Deletes a temporary BLOB or CLOB.
DBE_LOB.APPEND	Adds the content of a LOB to another LOB.
DBE_LOB.FILEOPEN	Opens a database-external file and returns a file descriptor.
DBE_LOB.FILECLOSE	Closes an external file opened by FILEOPEN .
DBE_LOB.LOADFROMFILE	Reads a database-external file to a BLOB file.
DBE_LOB.LOADBLOBFROMFILE	Reads a database-external file to a BLOB file.
DBE_LOB.LOADCLOBFROMFILE	Reads a database-external file to a CLOB file.
DBE_LOB.CONVERTTOBLOB	Converts a CLOB file to a BLOB file.
DBE_LOB.CONVERTTOCLOB	Converts a BLOB file to a CLOB file.

- [DBE_LOB.GET_LENGTH](#)

The stored procedure GET_LENGTH obtains and returns the size of a specified LOB.

The prototype of the DBE_LOB.GET_LENGTH function is as follows:

```
DBE_LOB.GET_LENGTH (  
lob IN BLOB)  
RETURN INTEGER;  
  
DBE_LOB.GET_LENGTH (  
lob IN CLOB)  
RETURN INTEGER;
```

Table 10-65 DBE_LOB.GET_LENGTH parameters

Parameter	Description
lob	BLOB/CLOB 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 dbe_lob.bfile,  
open_mode text DEFAULT 'null'::text  
);
```

Table 10-66 DBE_LOB.OPEN parameters

Parameter	Description
lob	BLOB or CLOB that is opened.

- DBE_LOB.READ

The stored procedure READ loads a part of LOB content to the buffer based on the specified length and initial position offset.

The prototype of the DBE_LOB.READ function is as follows:

```
DBE_LOB.READ (  
lob IN BLOB,  
len IN INTEGER,  
start IN INTEGER,  
buffer OUT RAW);  
  
DBE_LOB.READ (  
lob IN CLOB,  
len INOUT INTEGER,  
start IN INTEGER,  
buffer OUT VARCHAR2);
```

Table 10-67 DBE_LOB.READ parameters

Parameter	Description
lob	BLOB/CLOB to be read
len	Length of read content NOTE If the length is negative, the error message "ERROR: argument 2 is null, invalid, or out of range." is displayed.
start	Indicates where to start reading the LOB content, that is, the offset bytes to initial position of LOB content.
buffer	Target buffer to store the read LOB content.

- DBE_LOB.WRITE

The stored procedure WRITE copies content in the buffer to LOB variables based on the specified length and initial position.

The prototype of the DBE_LOB.WRITE function is as follows:

```
DBE_LOB.WRITE (  
dest_lob INOUT BLOB,  
len      IN    INTEGER,  
start    IN    INTEGER,  
src_lob  IN     RAW);  
  
DBE_LOB.WRITE (  
dest_lob INOUT CLOB,  
len      IN    INTEGER,  
start    IN    INTEGER,  
src_lob  IN     VARCHAR2);
```

Table 10-68 DBE_LOB.WRITE parameters

Parameter	Description
dest_lob	BLOB/CLOB to which the buffer content is written
len	Length to write, up to 32767 characters. NOTE If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.
start	Indicates where to start writing the LOB content, that is, the offset bytes to initial position of LOB content. NOTE If the offset value is less than 1, an error is reported. If the offset value is greater than the maximum length of LOB type contents, no error is reported.
src_lob	Content to be written

- DBE_LOB.WRITE_APPEND

The stored procedure WRITE_APPEND copies content in the buffer 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 (  
dest_lob INOUT BLOB,  
len IN INTEGER,  
src_lob IN RAW);  
  
DBE_LOB.WRITE_APPEND (  
dest_lob INOUT CLOB,  
len IN INTEGER,  
src_lob IN VARCHAR2);
```

Table 10-69 DBE_LOB.WRITE_APPEND parameters

Parameter	Description
dest_lob	BLOB/CLOB to which the buffer content is written
len	Length of written content. NOTE If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.
src_lob	Content to be written

- DBE_LOB.COPY

The stored procedure COPY copies content in a BLOB to another BLOB 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 10-70 DBE_LOB.COPY parameters

Parameter	Description
dest_lob	BLOB to be pasted.
src_lob	BLOB to be copied.
len	Length to copy NOTE If the length of copied content is shorter than 1 or longer than the maximum length of BLOB, an error is reported.
dest_start	Indicates where to start pasting the BLOB content, that is, the offset bytes to initial position of BLOB content. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB, an error is reported.

Parameter	Description
src_start	Indicates where to start copying the BLOB content, that is, the offset bytes to initial position of BLOB content. NOTE If the offset is shorter than 1 or longer than the length of source BLOB, an error is reported.

- DBE_LOB.ERASE

The stored procedure ERASE deletes content in BLOB based on the specified length and initial position offset.

The prototype of the DBE_LOB.ERASE function is as follows:

```
DBE_LOB.ERASE (  
lob      INOUT  BLOB,  
len      INOUT  INTEGER,  
start    IN     INTEGER DEFAULT 1);
```

Table 10-71 DBE_LOB.ERASE parameters

Parameter	Description
lob	BLOB whose content is to be deleted.
len	Length of content to be deleted. NOTE If the length of deleted content is shorter than 1 or longer than the maximum length of BLOB, an error is reported.
start	Indicates where to start deleting the BLOB content, that is, the offset bytes to initial position of BLOB content. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB, 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 10-72 DBE_LOB.CLOSE parameters

Parameter	Description
lob	BLOB/CLOB to be disabled.

- DBE_LOB.MATCH

This function returns the *N*th occurrence position of **pattern** in a LOB. **NULL** is returned for any of the following conditions: offset < 1 or offset > **LOBMAXSIZE**; *n*th < 1 or *n*th > **LOBMAXSIZE**.

The prototype of the DBE_LOB.MATCH function is as follows:

```
DBE_LOB.MATCH (
lob          IN   BLOB,
pattern      IN   RAW,
start_index  IN   INTEGER DEFAULT 1,
match_index  IN   INTEGER DEFAULT 1)
RETURN INTEGER;

DBE_LOB.MATCH (
lob          IN   CLOB,
pattern      IN   VARCHAR2,
start_index  IN   INTEGER DEFAULT 1,
match_index  IN   INTEGER DEFAULT 1)
RETURN INTEGER;
```

Table 10-73 DBE_LOB.MATCH API parameters

Parameter	Description
lob	BLOB/CLOB descriptor to be searched for.
pattern	Pattern to match. It is RAW for BLOB and TEXT for CLOB.
start_index	For BLOB, the absolute offset is in the unit of byte. For CLOB, the offset is in the unit of character. The matching start position is 1.
match_index	Number of pattern matching times. The minimum value is 1.

- DBE_LOB.COMPARE

This function compares two LOBs or a certain part of two LOBs.

- 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**.

The prototype of the DBE_LOB.COMPARE function is as follows:

```
DBE_LOB.COMPARE (
lob1  IN   BLOB,
lob2  IN   BLOB,
len   IN   INTEGER DEFAULT DBE_LOB.LOBMAXSIZE,
start1 IN   INTEGER DEFAULT 1,
start2 IN   INTEGER DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
lob1  IN   CLOB,
lob2  IN   CLOB,
len   IN   INTEGER DEFAULT DBE_LOB.LOBMAXSIZE,
start1 IN   INTEGER DEFAULT 1,
start2 IN   INTEGER DEFAULT 1)
RETURN INTEGER;
```

Table 10-74 DBE_LOB.COMPARE parameters

Parameter	Description
lob1	First BLOB/CLOB to be compared.
lob2	Second BLOB/CLOB to be compared.
len	Number of characters or bytes to be compared. The maximum value is DBE_LOB.LOBMAXSIZE .
start1	Offset of the first LOB descriptor. The initial position is 1.
start2	Offset of the second LOB descriptor. The initial position is 1.

- DBE_LOB.SUBSTR

This function reads the substring of a LOB and returns the number of read bytes or the number of characters. **NULL** is returned for any of the following conditions: amount > 1 or amount < 32767; offset < 1 or offset >

LOBMAXSIZE

The prototype of the DBE_LOB.SUBSTR function is as follows:

```
DBE_LOB.SUBSTR (  
lob      IN   BLOB,  
len      IN   INTEGER DEFAULT 32767,  
start    IN   INTEGER DEFAULT 1)  
RETURN RAW;
```

```
DBE_LOB.SUBSTR (  
lob      IN   CLOB,  
len      IN   INTEGER DEFAULT 32767,  
start    IN   INTEGER DEFAULT 1)  
RETURN VARCHAR2;
```

Table 10-75 DBE_LOB.SUBSTR parameters

Parameter	Description
lob	LOB descriptor of the substring to be read. For BLOB, the return value is the number of read bytes. For CLOB, the return value is the number of characters.
len	Number of bytes or characters to be read.
start	Number of characters or bytes offset from the start position.

- 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. If an empty LOB is truncated, no execution result is displayed. If the specified length is longer than the length of the LOB, an exception occurs.

The prototype of the DBE_LOB.STRIP function is as follows:

```
DBE_LOB.STRIP (  
lob      IN OUT   BLOB,
```

```
newlen IN INTEGER);  
  
DBE_LOB.STRIP (  
lob IN OUT CLOB,  
newlen IN INTEGER);
```

Table 10-76 DBE_LOB.STRIP parameters

Parameter	Description
lob	BLOB to be read.
newlen	New length after truncation, in bytes for BLOBs or in characters for CLOBs.

- 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 (  
locator INOUT BLOB,  
cache IN BOOLEAN,  
keep_alive_time IN INTEGER);  
  
DBE_LOB.CREATE_TEMPORARY (  
locator INOUT CLOB,  
cache IN BOOLEAN,  
keep_alive_time IN INTEGER);
```

Table 10-77 DBE_LOB.CREATE_TEMPORARY parameters

Parameter	Description
locator	LOB descriptor
cache	Used only for syntax compatibility.
keep_alive_time	Used only for syntax compatibility.

- DBE_LOB.APPEND

The stored procedure APPEND loads a part of BLOB content to the buffer based on the specified length and initial position offset.

The prototype of the DBE_LOB.APPEND function is as follows:

```
DBE_LOB.APPEND (  
dest_lob INOUT BLOB,  
src_lob IN BLOB);  
  
DBE_LOB.APPEND (  
dest_lob INOUT CLOB,  
src_lob IN CLOB);
```

Table 10-78 DBE_LOB.APPEND parameters

Parameter	Description
dest_lob	BLOB/CLOB to which the buffer content is to be written
src_lob	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 (  
lob_loc INOUT BLOB);  
  
DBE_LOB.FREETEMPORARY (  
lob_loc INOUT CLOB);
```

Table 10-79 DBE_LOB.FREETEMPORARY parameters

Parameter	Description
lob_loc	BLOB/CLOB to be freed.

- DBE_LOB.FILEOPEN

This function is used to open a database-external file of the BFILE type and return the file descriptor corresponding to the file.

The BFILE type is defined as follows:

```
DBE_LOB.BFILE (  
directory text,  
filename text);
```

The prototype of the DBE_LOB.FILEOPEN function is as follows:

```
DBE_LOB.FILEOPEN (  
file IN DBE_LOB.BFILE,  
open_mode IN text)  
RETURN integer;
```

Table 10-80 DBE_LOB.FILEOPEN parameters

Parameter	Description
file	Specifies the database-external file to be opened. The BFILE type contains the file path and file name.
open_mode	Specifies the file open mode (w , r , or a).

- 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 10-81 DBE_LOB.FILECLOSE parameters

Parameter	Description
file	External database file to be closed (that is opened by FILEOPEN).

- DBE_LOB.LOADFROMFILE

This is used to read an external BFILE file to a BLOB file.

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 10-82 DBE_LOB.LOADFROMFILE parameters

Parameter	Description
dest_lob	Target BLOB file. The BFILE file will be read to this file.
src_bfile	Source BFILE file to be read.
amount	Size of a BLOB file. If the size of a file exceeds this threshold, the file will not be saved to the BLOB file.
dest_offset	Offset length of the BLOB file. If dest_offset is set to 1 , data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the BFILE file. If src_offset is set to 1 , data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.LOADBLOBFROMFILE

This is used to read an external BFILE file to a BLOB file.

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 10-83 DBE_LOB.LOADBLOBFROMFILE parameters

Parameter	Description
dest_lob	Target BLOB file. The BFILE file will be read to this file.
src_bfile	Source BFILE file to be read.

Parameter	Description
amount	Size of a BLOB file. If the size of a file exceeds this threshold, the file will not be saved to the BLOB file.
dest_offset	Offset length of the BLOB file. If dest_offset is set to 1 , data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the BFILE file. If src_offset is set to 1 , data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.LOADCLOBFROMFILE

This is used to read an external BFILE file to a CLOB file.

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 10-84 DBE_LOB.LOADCLOBFROMFILE parameters

Parameter	Description
dest_lob	Target CLOB file. The BFILE file will be read to this file.
src_bfile	Source BFILE file to be read.
amount	Size of a CLOB file. If the size of a file exceeds this threshold, the file will not be saved to the CLOB file.
dest_offset	Offset length of the CLOB file. If dest_offset is set to 1 , data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the BFILE file. If src_offset is set to 1 , data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.CONVERTTOBLOB

This function is used to convert a CLOB file to a BLOB file.

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 10-85 DBE_LOB.CONVERTTOBLOB parameters

Parameter	Description
dest_lob	Target BLOB file, which is converted from a CLOB file.
src_bfile	Source CLOB file to be read.
amount	Size of a BLOB file. If the size of a file exceeds this threshold, the file will not be saved to the BLOB file.
dest_offset	Offset length of the BLOB file. If dest_offset is set to 1 , data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the CLOB file. If src_offset is set to 1 , data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.CONVERTTOCLOB

This function is used to convert a BLOB file to a CLOB file.

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 10-86 DBE_LOB.CONVERTTOCLOB parameters

Parameter	Description
dest_lob	Target CLOB file, which is converted from a BLOB file.
src_bfile	Source BLOB file to be read.
amount	Size of a CLOB file. If the size of a file exceeds this threshold, the file will not be saved to the CLOB file.
dest_offset	Offset length of the CLOB file. If dest_offset is set to 1 , data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the BLOB file. If src_offset is set to 1 , data is read from the start position of the file. The rest may be deduced by analogy.

Examples

```
-- Obtain the length of a string.  
SELECT DBE_LOB.GET_LENGTH('12345678');  
get_length  
-----  
8  
(1 row)
```

```
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');

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
```

10.12.2.2 DBE_RANDOM

APIs

Table 10-87 lists all APIs supported by the **DBE_RANDOM** package.

Table 10-87 DBE_RANDOM API parameters

API	Description
DBE_RANDOM.SET_SEED	Sets a seed for a random number.
DBE_RANDOM.GET_VALUE	Generates a random number between a specified low and a specified high .

- 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 10-88 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 10-89 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 min .
max	Sets the high bound for a random number. The generated random number is less than max .

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)
```

10.12.2.3 DBE_OUTPUT

APIs

[Table DBE_OUTPUT](#) provides all APIs supported by the **DBE_OUTPUT** package.

Table 10-90 DBE_OUTPUT

API	Description
DBE_OUTPUT.PRINT_LINE	Outputs the specified text with newline characters.
DBE_OUTPUT.PRINT	Outputs the specified text without newline characters.
DBE_OUTPUT.SET_BUFFER_SIZE	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.

- DBE_OUTPUT.PRINT_LINE

The stored procedure **PRINT_LINE** writes a row 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 10-91 DBE_OUTPUT.PRINT_LINE API parameters

Parameter	Description
format	Specifies the text that was written to the buffer.

- 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 10-92 DBE_OUTPUT.PRINT API parameters

Parameter	Description
format	Specifies the text that was written to the specified 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 10-93 DBE_OUTPUT.SET_BUFFER_SIZE API parameters

Parameter	Description
size	Sets the output buffer size.

Examples

```
BEGIN  
  DBE_OUTPUT.SET_BUFFER_SIZE(50);  
  DBE_OUTPUT.PRINT('hello, ');  
  DBE_OUTPUT.PRINT_LINE('database!');-- Output "hello, database!".  
END;  
/
```

10.12.2.4 DBE_RAW

APIs

[Table 10-94](#) provides all APIs supported by the **DBE_RAW** package.

Table 10-94 DBE_RAW

API	Description
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW	Converts a value of the INTEGER type to a binary representation (RAW type).
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER	Converts a binary representation (RAW type) to a value of the INTEGER type.

API	Description
DBE_RAW.GET_LENGTH	Obtains the length of a RAW object.
DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW	Converts a value of the VARCHAR2 type to a binary representation (RAW type).
DBE_RAW.CAST_TO_VARCHAR2	Converts a value of the RAW type to a value of the VARCHAR2 type.
DBE_RAW.SUBSTR	Returns the substring of the RAW type.
DBE_RAW.BIT_OR	Performs the bitwise OR operation on raw data.

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 10-95 `DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW` parameters

Parameter	Description
value	Specifies the <code>INTEGER</code> value to be converted to the RAW value.
endianess	Specifies the <code>INTEGER</code> value 1 or 2 for the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates 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,  
endianess  IN INTEGER DEFAULT 1)  
RETURN BINARY_INTEGER;
```

Table 10-96 DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER parameters

Parameter	Description
value	Specifies an INTEGER value in a binary representation (RAW type).
endianess	Specifies the INTEGER value 1 or 2 for the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates 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 10-97 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 10-98 DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW parameters

Parameter	Description
c	Specifies a VARCHAR2 object 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 10-99 DBE_RAW.CAST_TO_VARCHAR2 parameters

Parameter	Description
str	Specifies a RAW 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 RAW,  
  str2 IN RAW)  
RETURN RAW;
```

Table 10-100 DBE_RAW.BIT_OR parameters

Parameter	Description
str1	First character string of the bitwise OR operation
str2	Second character string of the bitwise OR operation

- 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 raw,  
  IN off_set integer default 1,  
  IN amount integer default 32767)  
RETURN RAW;
```

Table 10-101 DBE_RAW.SUBSTR parameters

Parameter	Description
lob_loc	Source raw character string
off_set	Start position of a substring. The default value is 1 .
amount	Length of a substring. The default value is 32767 .

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;  
/  
  
-- Call the stored procedure.  
CALL proc_raw();
```



```
-- Drop the stored procedure.  
DROP PROCEDURE proc_raw;
```

10.12.2.5 DBE_TASK

APIs

Table DBE_TASK lists all APIs supported by the **DBE_TASK** package.

Table 10-102 DBE_TASK

API	Description
DBE_TASK.SUBMIT	Commits a scheduled job. The job ID is automatically generated by the system.
DBE_TASK.JOB_SUBMIT	Same as DBE_TASK.SUBMIT . However, It provides syntax compatibility parameters.
DBE_TASK.ID_SUBMIT	Commits a scheduled job. The job ID is specified by the user.
DBE_TASK.CANCEL	Removes a scheduled task by job ID.
DBE_TASK.RUN	Executes a scheduled task.
DBE_TASK.FINISH	Disables or enables scheduled task execution.
DBE_TASK.UPDATE	Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval.
DBE_TASK.CHANGE	Same as DBE_TASK.UPDATE . However, It provides syntax compatibility parameters.
DBE_TASK.CONTENT	Modifies the job content attribute of a scheduled job.
DBE_TASK.NEXT_TIME	Modifies the next-execution time attribute of a scheduled task.
DBE_TASK.INTERVAL	Modifies the execution interval attribute of a scheduled task.

- **DBE_TASK.SUBMIT**

The stored procedure SUBMIT commits 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. This function can be called by using the CALL or SELECT statement. If you call this function by using the SELECT statement, 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 10-103 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 (sysdate). 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 sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left empty or set to null , 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:

```
openGauss=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1');
submit
-----
 31031
(1 row)

openGauss=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
submit
-----
   512
(1 row)

openGauss=# DECLARE
openGauss-#   jobid int;
openGauss-# BEGIN
openGauss$$   PERFORM DBE_TASK.SUBMIT('call pro_xxx();', sysdate, 'interval "5 minute"', jobid);
openGauss$$ END;
openGauss$$ /
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 10-104 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 db.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.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
next_date	Timestamp	IN	Yes	Specifies the next time the job will be executed. The default value is the current system time (sysdate). 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 sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left empty or set to null , 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 false , which is used only for syntax compatibility.
instance	Integer	IN	Yes	The default value is 0 , which is used only for syntax compatibility.
force	Boolean	IN	Yes	The default value is false , which is used only for syntax compatibility.

Example:

```
openGauss=# DECLARE
openGauss=#   id integer;
openGauss=# BEGIN
openGauss$$   id = DBE_TASK.JOB_SUBMIT(
openGauss$$     what => 'insert into t1 values (1, 2)',
openGauss$$     job_interval => 'sysdate + 1' --daily
openGauss$$   );
openGauss$$ END;
openGauss$/
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:

```
openGauss=# 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 10-105 DBE_TASK.CANCEL parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
id	Integer	IN	No	Specifies the job ID.

Example:

```
openGauss=# 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 10-106 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:

```
openGauss=# BEGIN
openGauss$#   DBE_TASK.ID_SUBMIT(12345, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
openGauss$#   DBE_TASK.RUN(12345);
openGauss$# END;
openGauss$# /
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 10-107 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, true for broken and false for not broken. The current job is updated based on the parameter value true or false . 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 broken is set to true , next_time is updated to '4000-1-1'. If broken is set to false and next_time is not empty, next_time is updated for the job. If next_time is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

Example:

```
openGauss=# CALL db_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');  
id_submit
```

```
-----  
(1 row)
```

```
openGauss=# CALL db_task.finish(101, true);  
finish
```

```
-----  
(1 row)
```

```
openGauss=# CALL db_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:

```
db_task.UPDATE(  
  id      IN  INTEGER,
```

```
content      IN TEXT,  
next_time    IN TIMESTAMP,  
interval_time IN TEXT);
```

Table 10-108 DBE_TASK.UPDATE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
id	Integer	IN	No	Specifies the job ID.
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 content parameter for the specified job. Otherwise, the system updates the content 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 next_time parameter for the specified job. Otherwise, the system updates the next_time 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 interval_time parameter for the specified job. Otherwise, the system updates the interval_time parameter for the specified job after necessary validity check. If this parameter is set to null , the job will be executed only once, and the job status will change to 'd' afterward.

Example:

```
openGauss=# CALL db_e_task.update(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');  
update  
-----  
(1 row)  
  
openGauss=# CALL db_e_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 10-109 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 what parameter for the specified job. Otherwise, the system updates the what 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 next_date parameter for the specified job. Otherwise, the system updates the next_date 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 job_interval parameter for the specified job. Otherwise, the system updates the job_interval parameter for the specified job after necessary validity check. If this parameter is set to null , the job will be executed only once, and the job status will change to 'd' afterward.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
instance	Integer	IN	Yes	Used only for syntax compatibility.
force	Boolean	IN	No	Used only for syntax compatibility.

Example:

```
openGauss=# BEGIN
openGauss## DBE_TASK.CHANGE(
openGauss##     job => 101,
openGauss##     what => 'insert into t2 values (2);'
openGauss## );
openGauss## END;
openGauss## /
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 10-110 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:

```
openGauss=# CALL db_task.content(101, 'call userproc();');
content
-----
(1 row)

openGauss=# CALL db_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 10-111 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:

```
openGauss=# CALL db_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 10-112 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 null , 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:

```
openGauss=# CALL dbe_task.interval(101, 'sysdate + 1.0/1440');
interval
-----
(1 row)

openGauss=# 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_NO...](#) 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.

- 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.
- 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.
- 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.

10.12.2.6 DBE_UTILITY

APIs

[Table 10-113](#) provides all APIs supported by the **DBE_UTILITY** package.

Table 10-113 DBE_UTILITY

API	Description
DBE_UTILITY.FORMAT_ERROR_BACKTRACE	Outputs the call stack of an abnormal stored procedure.
DBE_UTILITY.FORMAT_ERROR_STACK	Outputs detailed information about a stored procedure exception.
DBE_UTILITY.FORMAT_CALL_STACK	Outputs the call stack of a stored procedure.
DBE_UTILITY.GET_TIME	Outputs the current time, which is used to obtain the execution duration.

- 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;
```

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;  
/
```

10.12.2.7 DBE_SQL

APIs

[Table 10-114](#) lists APIs supported by the DBE_SQL package.

Table 10-114 DBE_SQL

API	Description
DBE_SQL.REGISTER_CONTEXT	Opens a cursor.
DBE_SQL.SQL_UNREGISTER_CONTEXT	Closes an open cursor.
DBE_SQL.SQL_SET_SQL	Passes a set of SQL statements to a cursor.
DBE_SQL.SQL_RUN	Performs a set of dynamically defined operations on a cursor.
DBE_SQL.NEXT_ROW	Reads a row of cursor data.
DBE_SQL.SET_RESULT_TYPE	Dynamically defines a column.

API	Description
DBE_SQL.SET_RESULT_TYPE_CHAR	Dynamically defines a column of the CHAR type.
DBE_SQL.SET_RESULT_TYPE_INT	Dynamically defines a column of the INT type.
DBE_SQL.SET_RESULT_TYPE_LONG	Dynamically defines a column of the LONG type.
DBE_SQL.SET_RESULT_TYPE_RAW	Dynamically defines a column of the RAW type.
DBE_SQL.SET_RESULT_TYPE_TEXT	Dynamically defines a column of the TEXT type.
DBE_SQL.SET_RESULT_TYPE_UNKOWN	Dynamically defines a column of an unknown type.
DBE_SQL.GET_RESULT	Reads a dynamically defined column value.
DBE_SQL.GET_RESULT_CHAR	Reads a dynamically defined column value of the CHAR type.
DBE_SQL.GET_RESULT_INT	Reads a dynamically defined column value of the INT type.
DBE_SQL.GET_RESULT_LONG	Reads a dynamically defined column value of the LONG type.
DBE_SQL.GET_RESULT_RAW	Reads a dynamically defined column value of the RAW type.
DBE_SQL.GET_RESULT_TEXT	Reads a dynamically defined column value of the TEXT type.
DBE_SQL.GET_RESULT_UNKNOWN	Reads a dynamically defined column value of an unknown type.
DBE_SQL.DBE_SQL_GET_RESULT_CHAR	Reads a dynamically defined column value of the CHAR type.
DBE_SQL.DBE_SQL_GET_RESULT_LONG	Reads a dynamically defined column value of the LONG type.
DBE_SQL.DBE_SQL_GET_RESULT_RAW	Reads a dynamically defined column value of the RAW type.
DBE_SQL.IS_ACTIVE	Checks whether a cursor is opened.
DBE_SQL.LAST_ROW_COUNT	Compatible API. This function is not supported currently.
DBE_SQL.RUN_AND_NEXT	Reserved API. This function is not supported currently.

API	Description
DBE_SQL.SQL_BIND_VARIABLE	Binds a value to a variable in a statement.
DBE_SQL.SQL_BIND_ARRAY	Binds a group of values to a variable in a statement.
DBE_SQL.SET_RESULT_TYPE_INTS	Dynamically defines a column of the INT array type.
DBE_SQL.SET_RESULT_TYPE_TEXTS	Dynamically defines a column of the TEXT array type.
DBE_SQL.SET_RESULT_TYPE_RAWS	Dynamically defines a column of the RAW array type.
DBE_SQL.SET_RESULT_TYPE_BYTES	Dynamically defines a column of the BYTEA array type.
DBE_SQL.SET_RESULT_TYPE_CHARS	Dynamically defines a column of the CHAR array type.
DBE_SQL.SET_RESULTS_TYPE	Dynamically defines a column of the array type.
DBE_SQL.GET_RESULTS_INT	Reads a dynamically defined column value of the INT array type.
DBE_SQL.GET_RESULTS_TEXT	Reads a dynamically defined column value of the TEXT array type.
DBE_SQL.GET_RESULTS_RAW	Reads a dynamically defined column value of the RAW array type.
DBE_SQL.GET_RESULTS_BYTEA	Reads a dynamically defined column value of the BYTEA array type.
DBE_SQL.GET_RESULTS_CHAR	Reads a dynamically defined column value of the CHAR array type.
DBE_SQL.GET_RESULTS	Reads a dynamically defined column value.
DBE_SQL.SQL_DESCRIBE_COLUMNS	Describes the column information read by the cursor.
DBE_SQL.DESC_REC	Stores the type of the column information read by the cursor.
DBE_SQL.DESC_TAB	TABLE type of DESC_REC.
DBE_SQL.DATE_TABLE	TABLE type of DATE.
DBE_SQL.NUMBER_TABLE	TABLE type of NUMBER.
DBE_SQL.VARCHAR2_TABLE	TABLE type of VARCHAR2.
DBE_SQL.BIND_VARIABLE	Binds parameters.

API	Description
DBE_SQL.SQL_SET_RESULTS_TYPE_C	Dynamically defines a column of the array type.
DBE_SQL.SQL_GET_VALUES_C	Reads a dynamically defined column value.
DBE_SQL.GET_VARIABLE_RESULT	Reads the return value of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_CHAR	Reads the return value (of the char type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_RAW	Reads the return value (of the raw type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_TEXT	Reads the return value (of the text type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_INT	Reads the return value (of the int type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_TEXT	Reads the return value (of the text array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_RAW	Reads the return value (of the raw array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_CHAR	Reads the return value (of the char array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_INT	Reads the return value (of the int array type) of an SQL statement.

 NOTE

- You are advised to use **dbe_sql.set_result_type** and **dbe_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 **dbe_sql** operations. This function does not transfer any parameter. It automatically generates cursor IDs in an ascending order and returns values to integer variables.
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 10-115 DBE_SQL.SQL_UNREGISTER_CONTEXT parameters

Parameter	Description
context_id	ID of the cursor to be closed.

- DBE_SQL.SQL_SET_SQL

This function is used to parse the query statement of a specified cursor. The statement parameters can be passed 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 10-116 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 number. Currently, only 1 is supported.

- DBE_SQL.SQL_RUN

This function executes a given cursor. It receives a cursor ID first, and the data obtained after execution is used for subsequent operations.

The prototype of the DBE_SQL.SQL_RUN function is as follows:

```
DBE_SQL.SQL_RUN(  
context_id IN INTEGER,  
)  
RETURN INTEGER;
```

Table 10-117 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 10-118 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 10-119 DBE_SQL.SET_RESULT_TYPE parameters

Parameter	Description
context_id	ID of the cursor to be executed.
pos	Position of a dynamically defined column in the query
column_ref	Variable of any type. You can select an appropriate API to dynamically define columns based on variable types.
maxsize	Length of a defined column.

- 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 10-120 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 10-121 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 10-122 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 10-123 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	Parameter of the RAW type.
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 10-124 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 10-125 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULT(  
context_id IN INTEGER,  
pos IN INTEGER,  
column_value INOUT ANYELEMENT  
)  
RETURN ANYELEMENT;
```

Table 10-126 DBE_SQL.GET_RESULT parameters

Parameter	Description
context_id	ID of the cursor to be executed.
pos	Position of a dynamically defined column in the query

Parameter	Description
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 function 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 10-127 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 number. It is an output parameter. The input parameter must be a variable. Currently, the output value is -1 regardless of the input parameter.
actual_length	Length of a return value.

The overloaded function of the DBE_SQL.GET_RESULT_CHAR function is as follows:

```
DBE_SQL.GET_RESULT_CHAR(  
context_id      IN  INTEGER,  
pos            IN  INTEGER,  
tr             INOUT CHARACTER  
);
```

Table 10-128 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.

- 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 10-129 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 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.

The prototype of the DBE_SQL.GET_RESULT_LONG function 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  
)  
RETURN RECORD;
```

Table 10-130 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 10-131 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 -1 regardless of the input parameter.
actual_length	Length of a return value. The value longer than this length will be truncated.

The overloaded function of the DBE_SQL.GET_RESULT_RAW function is as follows:

```
DBE_SQL.GET_RESULT_RAW(  
context_id      IN  INTEGER,  
pos            IN  INTEGER,  
tr             INOUT RAW  
);
```

Table 10-132 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.

- 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 10-133 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 10-134 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 10-135 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 10-136 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 10-137 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. When a cursor is opened, parsed, executed, or defined, the value is **TRUE**. When a cursor is closed, the value is **FALSE**. If the status is unknown, an error is reported. In other cases, the cursor is closed by default.

The prototype of the DBE_SQL.IS_ACTIVE function is as follows:

```
DBE_SQL.IS_ACTIVE(  
context_id          IN  INTEGER  
)  
RETURN BOOLEAN;
```

Table 10-138 DBE_SQL.IS_ACTIVE parameters

Parameter	Description
context_id	ID of the cursor to be queried.

- 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,  
language_flag in anyelement,  
out_value_size in int default null  
)  
RETURNS void;
```

Table 10-139 DBE_SQL.SQL_BIND_VARIABLE parameters

Parameter	Description
context_id	ID of the cursor to be queried.
query_string	Name of the bound variable
language_flag	Bound value
out_value_size	Size of the return value. Default value: null .

- 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(  
IN context_id int,  
IN query_string text,  
IN value anyarray  
)  
RETURNS void;  
DBE_SQL.SQL_BIND_ARRAY(  
IN context_id int,  
IN query_string text,  
IN value anyarray,  
IN lower_index int,  
IN higher_index int  
)  
RETURNS void;
```

Table 10-140 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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int  
)  
RETURNS integer;
```

Table 10-141 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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int,
```

```
    IN maxsize int  
)  
RETURNS integer;
```

Table 10-142 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.
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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int,  
    IN column_size int  
)  
RETURNS integer;
```

Table 10-143 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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int,  
    IN column_size int  
)  
RETURNS integer;
```

Table 10-144 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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int,  
    IN column_size int  
)  
RETURNS integer;
```

Table 10-145 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

Parameter	Description
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(  
    IN context_id int,  
    IN pos int,  
    IN column_ref anyarray,  
    IN cnt int,  
    IN lower_bnd int,  
    IN maxsize int DEFAULT 1024  
) returns void;
```

Table 10-146 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULTS_INT(  
    IN context_id int,  
    IN pos int,  
    INOUT column_value anyarray  
);
```

Table 10-147 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULTS_TEXT(  
    IN context_id int,  
    IN pos int,  
    INOUT column_value anyarray  
);
```

Table 10-148 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULTS_RAW(  
    IN context_id int,  
    IN pos int,  
    INOUT column_value anyarray  
);
```

Table 10-149 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULTS_BYTEA(  
    IN context_id int,  
    IN pos int,  
    INOUT column_value anyarray  
);
```

Table 10-150 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
column_value	Return value.

- DBE_SQL.GET_RESULTS_CHAR

This function 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 function is as follows:

```
DBE_SQL.GET_RESULTS_CHAR(  
    IN context_id int,  
    IN pos int,  
    INOUT column_value anyarray  
);
```

Table 10-151 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 function 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 function is as follows:

```
DBE_SQL.GET_RESULTS(  
    IN context_id int,  
    IN pos int,
```

```
INOUT column_value anyarray  
);
```

Table 10-152 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 db_sql.desc_tab  
)RETURNS record ;
```

Table 10-153 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.DESC_REC

This type is a composite type and is used to store the description of the SQL_DESCRIBE_COLUMNS API.

The prototype of the DBE_SQL.DESC_REC function is as follows:

```
CREATE TYPE DBE_SQL.DESC_REC AS (  
    col_type      int,  
    col_max_len   int,  
    col_name      VARCHAR2(32),  
    col_name_len  int,  
    col_schema_name VARCHAR2(32),  
    col_schema_name_len int,  
    col_precision int,  
    col_scale     int,  
    col_charsetid int,  
    col_charsetform int,  
    col_null_ok   BOOLEAN  
);
```

- DBE_SQL.DESC_TAB

This type is the TABLE type of DESC_REC and is implemented through the TABLE OF syntax.

The prototype of the DBE_SQL.DESC_TAB function is as follows:

```
CREATE TYPE DBE_SQL.DESC_TAB AS TABLE OF DBE_SQL.DESC_REC;
```

- DBE_SQL.DATE_TABLE

This type is the TABLE type of DATE and is implemented through the TABLE OF syntax.

The prototype of the DBE_SQL.DATE_TABLE function is as follows:

```
CREATE TYPE DBE_SQL.DATE_TABLE AS TABLE OF DATE;
```

- DBE_SQL.NUMBER_TABLE

This type is the TABLE type of NUMBER and is implemented through the TABLE OF syntax.

The prototype of the DBE_SQL.NUMBER_TABLE function is as follows:

```
CREATE TYPE DBE_SQL.NUMBER_TABLE AS TABLE OF NUMBER;
```

- DBE_SQL.VARCHAR2_TABLE

This type is the TABLE type of VARCHAR2 and is implemented through the TABLE OF syntax.

The prototype of the DBE_SQL.VARCHAR2 function is as follows:

```
CREATE TYPE DBE_SQL.VARCHAR2_TABLE AS TABLE OF VARCHAR2(2000);
```

- 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 10-154 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.

Parameter	Description
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 10-155 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 function is used to return the value of the bound OUT parameter and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_VARIABLE_RESULT function is as follows:

```
DBE_SQL.get_variable_result(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT column_value anyelement  
);
```

Table 10-156 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(  
    IN context_id int,  
    IN pos VARCHAR2  
)  
RETURNS char
```

Table 10-157 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 function is used to return the value of the bound OUT parameter of the RAW type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_VARIABLE_RESULT_RAW function is as follows:

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_raw(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT value anyelement  
)  
RETURNS anyelement
```

Table 10-158 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(  
    IN context_id int,  
    IN pos VARCHAR2  
)  
RETURNS text
```

Table 10-159 DBE_SQL.GET_VARIABLE_RESULT_TEXT parameters

Parameter	Description
context_id	ID of the cursor to be queried.

Parameter	Description
pos	Name of the bound parameter.

- DBE_SQL.GET_VARIABLE_RESULT_INT

This function is used to return the value of the bound OUT parameter of the INT type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_VARIABLE_RESULT_INT function is as follows:

```
DBE_SQL.get_variable_result_int(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT value anyelement  
)  
RETURNS anyelement
```

Table 10-160 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 function is used to return the value of the bound OUT parameter of the TEXT array type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_ARRAY_RESULT_TEXT function is as follows:

```
DBE_SQL.get_array_result_text(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT column_value anyarray  
)
```

Table 10-161 DBE_SQL.GET_ARRAY_RESULT_TEXT 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_RAW

This function is used to return the value of the bound OUT parameter of the RAW array type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_ARRAY_RESULT_RAW function is as follows:

```
DBE_SQL.get_array_result_raw(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT column_value anyarray  
)
```

Table 10-162 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 function is used to return the value of the bound OUT parameter of the CHAR array type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_ARRAY_RESULT_CHAR function is as follows:

```
DBE_SQL.get_array_result_char(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT column_value anyarray  
)
```

Table 10-163 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 function is used to return the value of the bound OUT parameter of the INT array type and obtain the OUT parameter in a stored procedure.

The prototype of the DBE_SQL.GET_ARRAY_RESULT_INT function is as follows:

```
DBE_SQL.get_array_result_int(  
    IN context_id int,  
    IN pos VARCHAR2,  
    INOUT column_value anyarray  
)
```

Table 10-164 DBE_SQL.GET_ARRAY_RESULT_INT parameters

Parameter	Description
context_id	ID of the cursor to be queried.
pos	Name of the bound parameter.

Parameter	Description
column_value	Return value.

Examples

```
-- Perform operations on RAW data in a stored procedure.
openGauss=# create or replace procedure pro_dbe_sql_all_02(in_raw raw,v_in int,v_offset int)
as
context_id int;
v_id int;
v_info bytea :=1;
query varchar(2000);
execute_ret int;
define_column_ret_raw bytea :='1';
define_column_ret int;
begin
drop table if exists pro_dbe_sql_all_tb1_02 ;
create table pro_dbe_sql_all_tb1_02(a int ,b blob);
insert into pro_dbe_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEF'));
insert into pro_dbe_sql_all_tb1_02 values(2,in_raw);
query := 'select * from pro_dbe_sql_all_tb1_02 order by 1';
-- Open a cursor.
context_id := dbe_sql.register_context();
-- Compile the cursor.
dbe_sql.sql_set_sql(context_id, query, 1);
-- Define columns.
define_column_ret:= dbe_sql.set_result_type(context_id,1,v_id);
define_column_ret_raw:= dbe_sql.set_result_type_raw(context_id,2,v_info,10);
-- Execute the cursor.
execute_ret := dbe_sql.sql_run(context_id);
loop
exit when (dbe_sql.next_row(cursoid) <= 0);
-- Obtain values.
dbe_sql.get_result(context_id,1,v_id);
dbe_sql.get_result_raw(context_id,2,v_info,v_in,v_offset);
-- Output the result.
dbe_output.print_line('id:'|| v_id || ' info:' || v_info);
end loop;
-- Close the cursor.
dbe_sql.sql_unregister_context(context_id);
end;
/
-- Call the stored procedure.
openGauss=# call pro_dbe_sql_all_02(HEXTORAW('DEADBEEF'),0,1);

-- Drop the stored procedure.
openGauss=# DROP PROCEDURE pro_dbe_sql_all_02;
```

10.12.2.8 DBE_FILE

The DBE_FILE package provides the capabilities of reading and writing OS text files for stored procedures.

APIs

[Table 10-165](#) lists all APIs supported by the DBE_FILE package.

Table 10-165 DBE_FILE

API	Description
DBE_FILE.OPEN	Opens a file based on the specified directory and file name.
DBE_FILE.IS_CLOSE	Checks whether a file is closed.
DBE_FILE.IS_OPEN	Checks whether a file is opened.
DBE_FILE.READ_LINE	Reads a line of a specified length from an open file handle.
DBE_FILE.WRITE	Writes data to an open file.
DBE_FILE.NEW_LINE	Writes one or more line terminators to an open file.
DBE_FILE.WRITE_LINE	Writes data to an open file and automatically appends a line terminator.
DBE_FILE.FORMAT_WRITE	Writes data in a specified format to the buffer of an open file.
DBE_FILE.GET_RAW	Reads RAW data from an open file.
DBE_FILE.PUT_RAW	Writes RAW data to an open file.
DBE_FILE.FLUSH	Writes cached data to a physical file.
DBE_FILE.CLOSE	Closes an open file handle.
DBE_FILE.CLOSE_ALL	Closes all file handles opened in a session.
DBE_FILE.REMOVE	Deletes a disk file. To perform this operation, you must have required permissions.
DBE_FILE.RENAME	Renames a disk file, which is similar to the mv command of Unix.
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.
DBE_FILE.GET_ATTR	Reads and returns the attributes of a disk file.
DBE_FILE.SEEK	Adjusts the position of a file pointer forward or backward based on the specified number of bytes.
DBE_FILE.GET_POS	Returns the current offset of a file, in bytes.

- [DBE_FILE.OPEN](#)

This function opens a file. You can specify the maximum number of characters in each line. A maximum of 50 files can be opened at a time. This function returns a handle of the INTEGER type.

The prototype of the DBE_FILE.OPEN function is as follows:

```
DBE_FILE.OPEN (  
dir          IN  VARCHAR2,  
file_name    IN  VARCHAR2,  
open_mode    IN  VARCHAR2,  
max_line_size IN  INTEGER DEFAULT 1024)  
RETURN INTEGER;
```

Table 10-166 DBE_FILE.OPEN API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dir	VARCHAR2	IN	No	Directory of a file. It is a string, indicating an object name. NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
file_name	VARCHAR2	IN	No	File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the OPEN function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).
open_mode	VARCHAR2	IN	No	File opening mode, including r (read text), w (write text), and a (append text). NOTE For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

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 1 and the maximum is 32767 . If this parameter is not specified, the default value 1024 is used.

- DBE_FILE.IS_OPEN

Checks whether a file 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;
```

Table 10-167 DBE_FILE.IS_OPEN parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	Yes	Handle of the file to be checked. If the handle is empty, DBE_FILE.IS_OPEN returns FALSE .

- 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;
```

Table 10-168 DBE_FILE.IS_CLOSE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	Yes	File handle to be detected.

- 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)
```

Table 10-169 DBE_FILE.READ_LINE 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.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
buffer	VARCHAR2	OUT	No	Buffer used to receive data
len	INTEGER	IN	Yes	Number of bytes read from a file. The default value is NULL . If the default value NULL is used, max_line_size 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;
```

Table 10-170 DBE_FILE.WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN. The file must be opened in write mode. This operation does not write a line terminator.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
buffer	TEXT	IN	Yes	Text data to be written to a file. The maximum buffer size is 32767 bytes. If no value is specified in the open state, the default value is 1024 bytes. Before the writing is performed, the buffer occupied by WRITE operations cannot exceed 32767 bytes. NOTE 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 := 1)  
RETURN BOOLEAN;
```

Table 10-171 DBE_FILE.NEW_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
line_nums	INTEGER	IN	Yes	Number of terminators written to a file. The default value is 1.

- 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;
```

Table 10-172 DBE_FILE.WRITE_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
buffer	TEXT	IN	Yes	Text data to be written to a file. The maximum buffer size is 32767 bytes. If no value is specified during the open state, the default value is 1024 bytes. Before the writing is performed, the buffer occupied by PUT operations cannot exceed 32767 bytes. NOTE 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 the disk after the writing.

- 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 VARCHAR2,  
arg1 IN VARCHAR2 DEFAULT NULL,  
...  
arg6 IN VARCHAR2 DEFAULT NULL])  
RETURN BOOLEAN;
```


Table 10-173 DBE_FILE.FORMAT_WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.
format	TEXT	IN	Yes	A string to be formatted, containing the text and format characters \n and %s.
[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

This function reads binary data from the opened file descriptor and returns the data using **r**.

The prototype of the DBE_FILE.GET_RAW function is as follows:

```
DBE_FILE.GET_RAW (  
file IN INTEGER,  
r OUT RAW,  
length IN INTEGER DEFAULT NULL)  
RETURN RAW;
```

Table 10-174 DBE_FILE.GET_RAW parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.
r	RAW	OUTPUT	No	Output binary data.
length	INTEGER	IN	Yes	Length of the file to be read. The default value is NULL . All data in the file is read. The maximum length is 1 GB.

- 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;
```

Table 10-175 DBE_FILE.PUT_RAW parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.
r	RAW	IN	No	Output binary data. NOTE 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 a file. The default value is false .

- 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;
```

Table 10-176 DBE_FILE.FLUSH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle

- 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.

The prototype of the DBE_FILE.CLOSE function is as follows:

```
DBE_FILE.CLOSE (  
file IN INTEGER  
)RETURN INTEGER;
```

Table 10-177 DBE_FILE.CLOSE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.

- 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;
```

Table 10-178 DBE_FILE.CLOSE_ALL API parameters

Parameter	Description
None.	None.

- 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  VARCHAR2,  
  file_name IN  VARCHAR2)  
RETURN VOID;
```

Table 10-179 DBE_FILE.REMOVE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dir	VARCHAR2	IN	No	Directory of a file. It is a string, indicating an object name. NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
file_name	VARCHAR2	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  VARCHAR2,  
src_file_name IN  VARCHAR2,  
dest_dir     IN  VARCHAR2,  
dest_file_name IN VARCHAR2,  
overwrite   IN  BOOLEAN DEFAULT FALSE)  
RETURN VOID;
```

Table 10-180 DBE_FILE.RENAME parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
src_dir	VARCHAR2	IN	No	Directory of the original file (case-sensitive) NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
src_file_name	VARCHAR2	IN	No	Original file to be renamed
dest_dir	VARCHAR2	IN	No	Target directory (case-sensitive). NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_file_name	VARCHAR2	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   VARCHAR2,  
src_file_name IN   VARCHAR2,  
dest_dir     IN   VARCHAR2,  
dest_file_name IN  VARCHAR2,  
start_line   IN   INTEGER DEFAULT 1,  
end_line     IN   INTEGER DEFAULT NULL)  
RETURN VOID;
```

Table 10-181 DBE_FILE.COPY parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
src_dir	VARCHAR2	IN	No	Directory of the original file NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
src_file_name	VARCHAR2	IN	No	File to be copied.
dest_dir	VARCHAR2	IN	No	Directory of the destination file NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
dest_file_name	VARCHAR2	IN	No	Target file NOTE For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
start_line	INTEGER	IN	No	Number of the line where the copy starts. The default value is 1 .
end_line	INTEGER	IN	Yes	Number of the line where the copy ends. The default value is NULL , 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,
OUT fexists boolean,
OUT file_length bigint,
OUT block_size integer);
```

Table 10-182 DBE_FILE.GET_ATTR parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
location	TEXT	IN	No	File directory NOTE - File directories need to be added to the system catalog PG_DIRECTORY. If the input path does not match the path in PG_DIRECTORY, an error indicating that the path does not exist will be reported. Functions that involve location as parameters also comply with this rule. - When the GUC parameter safe_data_path is enabled, you can only use an advanced package to read and write files in the file path specified by safe_data_path.
filename	TEXT	IN	No	File name.
file_exists	BOOLEAN	OUT	No	Specifies whether the file exists.
file_length	BIGINT	OUT	No	File length (unit: byte). If the file does not exist, NULL is returned.
block_size	INTEGER	OUT	No	Block size of the file system (unit: byte). If the file does not exist, NULL 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;
```

Table 10-183 DBE_FILE.SEEK parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.
absolute_start	BIGINT	IN	Yes	Absolute offset of a file. The default value is NULL .
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 NULL . If both absolute_start and this parameter are specified, the absolute_start 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;
```

Table 10-184 DBE_FILE.GET_POS parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened using OPEN.

Examples

```
-- Add the /temp/ directory to the PG_DIRECTORY system catalog as a SYSADMIN.
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 dbe_file.write(f, 'B');
  PERFORM dbe_file.new_line(f);
  PERFORM dbe_file.format_write(f, '[1 -> %s, 2 -> %s, 3 -> %s, 4 -> %s, 5 -> %s]', 'gaussdb', 'dbe', 'file',
'get', 'line');
  PERFORM dbe_file.new_line(f);
  PERFORM dbe_file.write_line(f, '1234567890');
  f := dbe_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 := dbe_file.open(dir, 'sample.txt', 'r');
  FOR i IN 1..11 LOOP
    RAISE INFO '[%] : %', i, dbe_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 := dbe_file.open(dir => dir, file_name => 'sample.txt', open_mode => 'R');
    dbe_output.print_line('before seek: current position is ' || dbe_file.get_pos(file => l_file)); -- before seek:
current position is 0
    dbe_file.seek(file => l_file, absolute_start=>abs_offset, relative_start=>rel_offset);
    dbe_output.print_line('fseek: current position is ' || dbe_file.get_pos(file => l_file)); -- seek: current
position is 100
    l_file := dbe_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
```

10.12.2.9 DBE_SESSION

APIs

Table 10-185 provides all APIs supported by the **DBE_SESSION** package. **DBE_SESSION** takes effect at the session level.

Table 10-185 DBE_SESSION

API	Description
DBE_SESSION.SET_CONTEXT	Sets the value of an attribute in a specified context.
DBE_SESSION.CLEAR_CONTEXT	Clears the value of an attribute in a specified context.
DBE_SESSION.SEARCH_CONTEXT	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 10-186 DBE_SESSION.SET_CONTEXT API parameters

Parameter	Description
namespace	Name of the context to be set. If the context does not exist, create a context. The value contains a maximum of 128 characters.
attribute	Attribute name. The value contains a maximum of 128 characters.
value	Name of the value to be set. The value contains a maximum of 128 characters.

- **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  
)returns void ;
```

Table 10-187 DBE_SESSION.CLEAR_CONTEXT API parameters

Parameter	Description
namespace	User-specified context.
client_identifier	Client authentication. The default value is ' null '. Generally, you do not need to manually set this parameter.
attribute	Attribute to be 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 10-188 DBE_SESSION.SEARCH_CONTEXT API parameters

Parameter	Description
namespace	User-specified context.
attribute	Attribute to be queried.

Example

```
BEGIN
  select DBE_SESSION.set_context('test', 'gaussdb', 'one'); -- Set the gaussdb attribute of the test context
to one.
  select DBE_SESSION.search_context('test', 'gaussdb');
  select DBE_SESSION.clear_context('test', 'test', 'gaussdb');
END;
/
```

10.12.2.10 DBE_MATCH

API Description

[Table 10-189](#) provides all APIs supported by the **DBE_MATCH** package.

Table 10-189 DBE_MATCH

API	Description
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.

- 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 10-190 DBE_MATCH.EDIT_DISTANCE_SIMILARITY API parameters

Parameter	Description
str1	First character string. If the value is null , 0 is returned.
str2	Second character string. If the value is null , 0 is returned.

10.12.2.11 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 10-191](#).

NOTICE

DBE_SCHEDULER does not support scheduled jobs for synchronizing data between nodes. To create scheduled jobs for multiple nodes, use [DBE_TASK](#).

Table 10-191 DBE_SCHEDULER

API	Description
DBE_SCHEDULER.CREATE_JOB	Creates a scheduled job.
DBE_SCHEDULER.DROP_JOB	Deletes a scheduled job.
DBE_SCHEDULER.DROP_SINGLE_JOB	Deletes a single scheduled job.
DBE_SCHEDULER.SET_ATTRIBUTE	Sets object attributes.
DBE_SCHEDULER.RUN_JOB	Executes a scheduled job.
DBE_SCHEDULER.RUN_BACKEND_JOB	Runs a scheduled job in the backend.
DBE_SCHEDULER.RUN_FRONTEND_JOB	Runs a scheduled job in the frontend.
DBE_SCHEDULER.STOP_JOB	Stops a scheduled job.
DBE_SCHEDULER.STOP_SINGLE_JOB	Stops a single scheduled job.

API	Description
DBE_SCHEDULER.GENERATE_JOB_NAME	Generates the name of a scheduled job.
DBE_SCHEDULER.CREATE_PROGRAM	Creates a program.
DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT	Defines program parameters.
DBE_SCHEDULER.DROP_PROGRAM	Deletes a program.
DBE_SCHEDULER.DROP_SINGLE_PROGRAM	Deletes a single program.
DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE	Sets the parameters of a scheduled job.
DBE_SCHEDULER.CREATE_SCHEDULE	Creates a schedule.
DBE_SCHEDULER.DROP_SCHEDULE	Deletes a schedule.
DBE_SCHEDULER.DROP_SINGLE_SCHEDULE	Deletes a single schedule.
DBE_SCHEDULER.CREATE_JOB_CLASS	Creates the class of a scheduled job.
DBE_SCHEDULER.DROP_JOB_CLASS	Deletes the class of a scheduled job.
DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS	Deletes the class of a single scheduled job.
DBE_SCHEDULER.GRANT_USER_AUTHORIZATION	Grants special permissions to a user.
DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION	Revokes special permissions from a user.
DBE_SCHEDULER.CREATE_CREDENTIAL	Creates a certificate.
DBE_SCHEDULER.DROP_CREDENTIAL	Destroys a certificate.
DBE_SCHEDULER.ENABLE	Enables an object.
DBE_SCHEDULER.ENABLE_SINGLE	Enables a single object.
DBE_SCHEDULER.DISABLE	Disables an object.

API	Description
DBE_SCHEDULER.DISABLE_SINGLE	Disables a single object.
DBE_SCHEDULER.EVAL_CALENDAR_STRING	Analyzes the scheduling job period.
DBE_SCHEDULER.EVALUATE_CALENDAR_STRING	Analyzes the scheduling job period.

- 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.

For the create API, the validity of input parameter types is not verified. Successful creation does not mean successful execution. You can query the execution status of the current job in the pg_job system catalog.

Table 10-192 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: • 'PLSQL_BLOCK': fast anonymous stored procedure. • 'STORED_PROCEDURE': stored procedure that is saved. • 'EXTERNAL_SCRIPT': external script.
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.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
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.
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 ' REGULAR ' 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:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select  
pg_sleep(1);', 3, false, 'test');  
create_program
```

(1 row)

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
```

```

create_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
create_job
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_schedule
-----
(1 row)

openGauss=# 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

DBE_SCHEDULER.DROP_JOB can specify one or more jobs, or specify a job class to delete a scheduled job.

Table 10-193 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. • true: The current scheduled job is stopped and then deleted. • false: The scheduled job fails to be deleted if it is running.
defer	boolean	IN	No	Specifies whether to delete a scheduled job. • true: A scheduled job can be deleted after it is complete. • false: The scheduled job cannot be executed and is deleted.
commit_semantics	text	IN	No	Commit rules: • 'STOP_ON_FIRST_ERROR': The deletion operation performed before the first error is reported is committed. • 'TRANSACTIONAL': Transaction-level commit. The deletion operation performed before an error is reported will be rolled back. • 'ABSORB_ERRORS': The system attempts to bypass an error and commit the deletion operation that is performed successfully.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
create_program
```

(1 row)

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
create_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
create_job
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_schedule
-----
(1 row)

openGauss=# 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
)
```

Table 10-194 DBE_SCHEDULER.DROP_SINGLE_JOB API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Specifies the name of a scheduled job or scheduled job class.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
force	boolean	IN	No	Specifies whether to delete a scheduled job. • true: The current scheduled job is stopped and then deleted. • false: The scheduled job fails to be deleted if it is running.
defer	boolean	IN	No	Specifies whether to delete a scheduled job. • true: A scheduled job can be deleted after it is complete. • false: The scheduled job cannot be executed and is deleted.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 0, false, 'test');
create_program
```

(1 row)

```
openGauss=# 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)

```
openGauss=# CALL DBE_SCHEDULER.drop_single_job('job1', false, false);
drop_single_job
```

(1 row)

```
openGauss=# 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 10-195 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.
value	boolean/date/timestamp/timestamp with time zone/text	IN	No	Attribute value. The options are as follows: 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. Scheduling-related: program_action, program_type, number_of_arguments, comments. Program-related: start_date, repeat_interval, end_date, comments.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
value2	text	IN	Yes	Additional attribute value. Reserved parameter bit. Currently, the target attribute with extra attribute values is not supported.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
create_program
```

(1 row)

```
openGauss=# CALL DBE_SCHEDULER.set_attribute('program1', 'number_of_arguments', 0);
set_attribute
```

(1 row)

```
openGauss=# CALL DBE_SCHEDULER.set_attribute('program1', 'program_type',
'STORED_PROCEDURE');
set_attribute
```

(1 row)

```
openGauss=# 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 10-196 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. • true: Use the current session to check whether the scheduled job can run properly. • false: Start the scheduled job in the backend. The execution result is recorded in logs.

Example:

```
openGauss=# 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)

```
openGauss=# CALL DBE_SCHEDULER.run_job('job1', false);
run_job
```

(1 row)

```
openGauss=# 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  
)
```

Table 10-197 DBE_SCHEDULER.RUN_BACKEND_JOB API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled job.

Example:

```
openGauss=# 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)

```
openGauss=# CALL DBE_SCHEDULER.run_backend_job('job1');  
run_backend_job
```

(1 row)

```
openGauss=# 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
```

Table 10-198 DBE_SCHEDULER.RUN_FOREGROUND_JOB API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled job.

Example:

```
openGauss=# create user test1 identified by '*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# select DBE_SCHEDULER.create_credential('cre_1', 'test1', '*****');
create_credential
-----
(1 row)

openGauss=# 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)

openGauss=# CALL DBE_SCHEDULER.run_foreground_job('job1');
run_foreground_job
-----
Host key verification failed.\r+
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
drop_credential
-----
(1 row)

openGauss=# 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 10-199 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 (.).
force	boolean	IN	No	Specifies whether to delete a scheduled job. • true: The scheduler sends a termination signal to end the job thread immediately. • false: The scheduler attempts to use the interrupt signal to terminate the scheduled job thread.
commit_semantics	text	IN	No	Commit rules: • 'STOP_ON_FIRST_ERROR': The interrupt operation performed before the first error is reported is committed. • 'ABSORB_ERRORS': The system attempts to bypass an error and commit the interrupt operation that is performed successfully.

Example:

```
openGauss=# 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)
```

```
openGauss=# CALL DBE_SCHEDULER.stop_job('job1', true, 'STOP_ON_FIRST_ERROR');
stop_job
```

```
-----
(1 row)
```

```
openGauss=# 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  
)
```

Table 10-200 DBE_SCHEDULER.STOP_SINGLE_JOB API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Specifies the name of a scheduled job or scheduled job class.
force	boolean	IN	No	Specifies whether to delete a scheduled job. true: The scheduler sends a termination signal to end the job thread immediately. false: The scheduler attempts to use the interrupt signal to terminate the scheduled job thread.

Example:

```
openGauss=# 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)  
  
openGauss=# CALL DBE_SCHEDULER.stop_single_job('job1', true);  
stop_single_job  
-----  
(1 row)  
  
openGauss=# 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 10-201 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: job\$_1 , job\$_2 , job\$_3 , ...

Example:

```
openGauss=# CALL DBE_SCHEDULER.generate_job_name();  
generate_job_name  
-----  
JOB$_1  
(1 row)  
  
openGauss=# CALL DBE_SCHEDULER.generate_job_name();  
generate_job_name  
-----  
JOB$_2  
(1 row)  
  
openGauss=# CALL DBE_SCHEDULER.generate_job_name('job');  
generate_job_name  
-----  
job3  
(1 row)  
  
openGauss=# 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  
)
```

Table 10-202 DBE_SCHEDULER.CREATE_PROGRAM API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Program name.
program_type	text	IN	No	Program type. The options are as follows: • 'PLSQL_BLOCK': fast anonymous stored procedure. • 'STORED_PROCEDURE': stored procedure that is saved. • 'EXTERNAL_SCRIPT': external script.
program_action	text	IN	No	Program operation.
number_of_arguments	integer	IN	No	Number of parameters used by the program.
enabled	boolean	IN	No	Specifies whether the program is enabled.
comments	text	IN	Yes	Comments.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select  
pg_sleep(1);', 3, false, 'test');  
create_program
```

(1 row)

```
openGauss=# 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  
)
```

Table 10-203 DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Program name.
argument_position	integer	IN	No	Parameter location.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
argument_name	text	IN	No	Parameter name.
argument_type	text	IN	No	Parameter type.
default_value	text	IN	No	Default value.
out_argument	boolean	IN	No	Reserved parameter.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', false);
define_program_argument
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', 'value1',
false);
define_program_argument
-----
(1 row)

openGauss=# 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
)
```

Table 10-204 DBE_SCHEDULER.DROP_PROGRAM API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Program name.
force	boolean	IN	No	Specifies whether to delete a program. • true: Before a program is deleted, all jobs that use the program are disabled. • false: The program cannot be referenced by any job. Otherwise, an error is sent.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
```

(1 row)

```
openGauss=# 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
)
```

Table 10-205 DBE_SCHEDULER.DROP_SINGLE_PROGRAM API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Program name.
force	boolean	IN	No	Specifies whether to delete a program. • true: Before a program is deleted, all jobs that use the program are disabled. • false: The program cannot be referenced by any job. Otherwise, an error is sent.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
```

```
-----
(1 row)
```

```
openGauss=# 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
)
```

Table 10-206 DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE API parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled job.
argument_position	integer	IN	No	Parameter location.
argument_name	text	IN	Yes	Parameter name.
argument_value	text	IN	Yes	Parameter value.

Example:

```
openGauss=# CALL db_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)

```
openGauss=# CALL DBE_SCHEDULER.set_job_argument_value('job1', 1, 'value1');
set_job_argument_value
```

(1 row)

```
openGauss=# 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
)
```

Table 10-207 DBE_SCHEDULER.CREATE_SCHEDULE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
start_date	timestamp with time zone	IN	Yes	Start time of a schedule.
repeat_interval	text	IN	No	Repetition frequency of a schedule.
end_date	timestamp with time zone	IN	Yes	End time of a schedule.
comments	text	IN	Yes	Comments.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)', null, 'test1');
create_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;', null, 'test1');
create_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
create_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
drop_single_schedule
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
drop_single_schedule
-----
(1 row)

openGauss=# 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  
)
```

Table 10-208 DBE_SCHEDULER.DROP_SCHEDULE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
force	boolean	IN	No	Specifies whether to delete a schedule. true: Any jobs or windows that use this schedule are disabled before the schedule is deleted. false: The schedule cannot be referenced by any job or window. Otherwise, an error occurs.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)', null, 'test1');  
create_schedule  
-----  
(1 row)  
  
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;', null, 'test1');  
create_schedule  
-----  
(1 row)  
  
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');  
create_schedule  
-----  
(1 row)  
  
openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');  
drop_single_schedule  
-----
```



```
(1 row)
```

```
openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
drop_single_schedule
-----
```

```
(1 row)
```

```
openGauss=# 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
)
```

Table 10-209 DBE_SCHEDULER.DROP_SINGLE_SCHEDULE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
force	boolean	IN	No	Specifies whether to delete a schedule. • true: Any jobs or windows that use this schedule are disabled before the schedule is deleted. • false: The schedule cannot be referenced by any job or window. Otherwise, an error occurs.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)', null, 'test1');
create_schedule
-----
```

```
(1 row)
```

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;', null, 'test1');
```

```
create_schedule
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY;
BYHOUR=6;');
create_schedule
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_schedule('schedule1');
drop_single_schedule
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_schedule('schedule2', false);
drop_single_schedule
-----

(1 row)

openGauss=# 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
)
```

Table 10-210 DBE_SCHEDULER.CREATE_JOB_CLASS API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled job class.
resource_consumer_group	text	IN	Yes	Inline resource consumer group of a scheduled job class.
service	text	IN	Yes	Inline database service of a scheduled job class.
logging_level	integer	IN	No	Number of scheduled job records.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
log_history	integer	IN	Yes	Number of days for storing scheduled job records.
comments	text	IN	Yes	Comments.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',  
resource_consumer_group => '123');  
create_job_class
```

```
-----  
(1 row)
```

```
openGauss=# 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  
)
```

Table 10-211 DBE_SCHEDULER.DROP_JOB_CLASS API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled job class.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
force	boolean	IN	No	Specifies whether to delete a scheduled job class. • true: Jobs of this class will be disabled, and another class will be set as the default class. Only when such setting is successful, this class will be deleted. • false: The class to be deleted cannot be referenced by any job. Otherwise, an error occurs.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',  
resource_consumer_group => '123');  
create_job_class
```

(1 row)

```
openGauss=# 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  
)
```

Table 10-212 DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled job class.
force	boolean	IN	No	Specifies whether to delete a scheduled job class. • true: Jobs of this class will be disabled, and another class will be set as the default class. Only when such setting is successful, this class will be deleted. • false: The class to be deleted cannot be referenced by any job. Otherwise, an error occurs.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',
resource_consumer_group => '123');
create_job_class
```

(1 row)

```
openGauss=# 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
)
```

Table 10-213 DBE_SCHEDULER.GRANT_USER_AUTHORIZATION API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
username	text	IN	No	Database username.
privilege	text	IN	No	Scheduled job permission.

Example:

```
openGauss=# create user user1 password '1*G*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
grant_user_authorization
-----
```

```
(1 row)
openGauss=# 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
)
```

Table 10-214 DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
username	text	IN	No	Database username.
privilege	text	IN	No	Scheduled job permission.

Example:

```
openGauss=# create user user1 password '1*G*****';
NOTICE: The encrypted password contains MD5 ciphertext, which is not secure.
```

```
CREATE ROLE
openGauss=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
grant_user_authorization
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.revoke_user_authorization('user1', 'create job');
revoke_user_authorization
-----
(1 row)

openGauss=# 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
)
```

Table 10-215 DBE_SCHEDULER.CREATE_CREDENTIAL API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
credential_name	text	IN	No	Name of the authorization certificate.
username	text	IN	No	Database username.
password	text	IN	Yes	User password.
database_role	text	IN	Yes	Database system permission.
windows_domain	text	IN	Yes	Domain to which a Windows user belongs.
comments	text	IN	Yes	Comments.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
create_credential
-----
(1 row)
```

```
openGauss=# 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
)
```

Table 10-216 DBE_SCHEDULER.DROP_CREDENTIAL API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
credential_name	text	IN	No	Name of the authorization certificate.
force	boolean	IN	No	Specifies whether to delete the authorization certificate. true: The certificate will be deleted no matter whether it is referenced by any job. false: No job can reference the certificate. Otherwise, an error occurs.

Example:

```
openGauss=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
create_credential
-----
```



```
(1 row)
```

```
openGauss=# 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'  
)
```

Table 10-217 DBE_SCHEDULER.ENABLE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name. You can specify one or more objects. If you specify multiple objects, separate them with commas (,).
commit_semantics	text	IN	No	Commit rules. The following types are supported: • 'STOP_ON_FIRST_ERROR': The enabling operation performed before the first error is reported is committed. • 'TRANSACTIONAL': Transaction-level commit. The enabling operation performed before an error is reported will be rolled back. • 'ABSORB_ERRORS': The system attempts to bypass an error and commit the enabling operation that is performed successfully.

Example:

```
openGauss=# CALL 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);  
lse, 'test');  
create_job
```

```
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
create_program
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.enable('job1');
enable
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.enable('program1', 'STOP_ON_FIRST_ERROR');
enable
-----

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
-----

(1 row)

openGauss=# 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
)
```

Table 10-218 DBE_SCHEDULER.ENABLE_SINGLE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.

Example:

```
openGauss=# 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);
lse, 'test');
create_job
```

```
-----  
(1 row)
```

```
openGauss=# CALL DBE_SCHEDULER.enable_single('job1');  
enable_single
```

```
-----  
(1 row)
```

```
openGauss=# 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'  
)
```

Table 10-219 DBE_SCHEDULER.DISABLE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.
force	boolean	IN	No	Specifies whether to disable an object. • true: The object is disabled regardless of whether other objects depend on it. • false: No object can depend on the object. Otherwise, an error occurs.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
commit_semantics	text	IN	No	Commit rules. The following types are supported: • 'STOP_ON_FIRST_ERROR': The disabling operation performed before the first error is reported is committed. • 'TRANSACTIONAL': Transaction-level commit. The disabling operation performed before an error is reported will be rolled back. • 'ABSORB_ERRORS': The system attempts to bypass an error and commit the disabling operation that is performed successfully.

Example:

```

openGauss=# 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)

openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
create_program
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.disable('job1');
disable
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.disable('program1', false, 'STOP_ON_FIRST_ERROR');
disable
-----
(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
drop_job
-----
(1 row)

```

```
openGauss=# 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
)
```

Table 10-220 DBE_SCHEDULER.DISABLE_SINGLE API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.
force	boolean	IN	No	Specifies whether to disable an object. • true: The object is disabled regardless of whether other objects depend on it. • false: No object can depend on the object. Otherwise, an error occurs.

Example:

```
openGauss=# CALL 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)

```
openGauss=# CALL DBE_SCHEDULER.disable_single('job1', false);
disable_single
-----
```

(1 row)

```
openGauss=# 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
```

Table 10-221 DBE_SCHEDULER.EVAL_CALENDAR_STRING API parameters

Parameter	Type	Input/ Output Param eter	Can Be Empty	Description
calendar_string	text	IN	No	Date string of a scheduled job.
start_date	timestamp with time zone	IN	No	Start time of a scheduled job.
return_date_after	timestamp with time zone	IN	No	Date when a scheduled job is returned.

Example:

```
openGauss=# CALL DBE_SCHEDULER.eval_calendar_string('FREQ=DAILY; BYHOUR=6;', sysdate,  
eval_calendar_string  
-----  
2023-09-15 06:47:24+08  
(1 row)
```

- **DBE_SCHEDULER.EVALUATE_CALENDAR_STRING**

Analyzes the scheduling job period.

Return type: timestamp with time zone

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
```

Table 10-222 DBE_SCHEDULER.EVALUATE_CALENDAR_STRING API parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
calendar_string	text	IN	No	Date string of a scheduled job.
start_date	timestamp with time zone	IN	No	Start time of a scheduled job.
return_date_after	timestamp with time zone	IN	No	Date when a scheduled job is returned.
next_run_date	timestamp with time zone	OUT	No	The next date when a scheduled job is returned.

Example:

```
openGauss=# 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

openGauss=# 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)
```

10.12.2.12 DBE_APPLICATION_INFO

APIs

[Table 10-223](#) provides all APIs supported by the **DBE_APPLICATION_INFO** package. **DBE_APPLICATION_INFO** applies to the current session.

Table 10-223 DBE_APPLICATION_INFO

API	Description
DBE_APPLICATION_INFO.SET_CLIENT_INFO	Writes client information.
DBE_APPLICATION_INFO.READ_CLIENT_INFO	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 10-224 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 10-225 DBE_APPLICATION_INFO.READ_CLIENT_INFO API parameters

Parameter	Description
client_info	Client information

10.13 Retry Management

Retry is a process in which the database executes an 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:

```
openGauss=# 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;
/
openGauss=# CALL retry_basic(1);
```

10.14 Debugging

Syntax

RAISE has the following five syntax formats:

Figure 10-34 raise_format::=

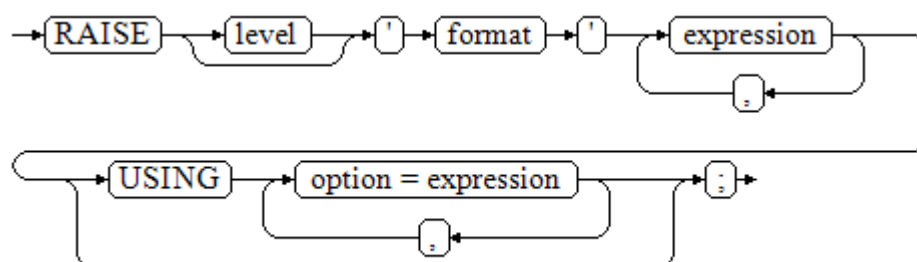


Figure 10-35 raise_condition::=

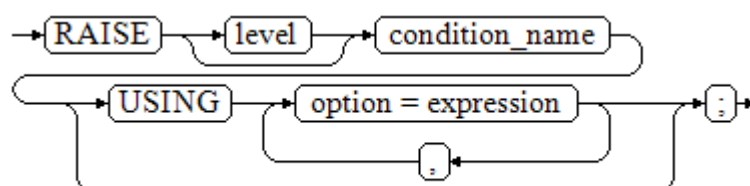


Figure 10-36 raise_sqlstate::=

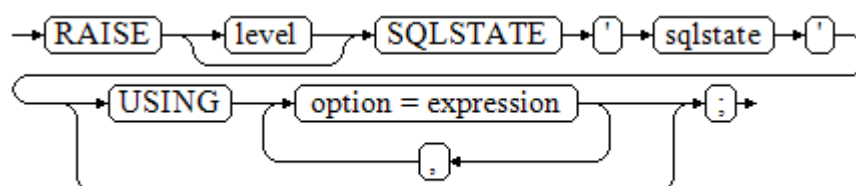


Figure 10-37 raise_option::=

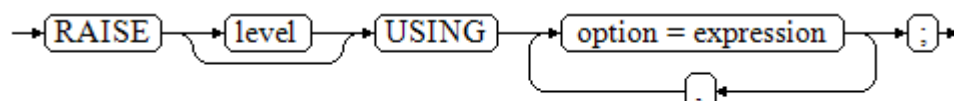
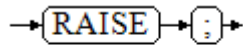


Figure 10-38 raise::=**Parameter description:**

- 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 parameters **log_min_messages** and **client_min_messages** determine 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 main transaction. Committing and rolling back an autonomous transaction does not affect the data that has been committed by the main transaction. In addition, an autonomous transaction is not affected by the main 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.
create table t2(a int, b int);
insert into t2 values(1,2);
select * from t2;

-- Create a stored procedure that contains an autonomous transaction.
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. ');
END;
/

-- Create a common stored procedure that calls an autonomous transaction stored procedure.
CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int) AS
DECLARE
BEGIN
    db_output.print_line('just no use call. ');
    insert into t2 values(666, 666);
    autonomous_4(a,b);
    rollback;
END;
```

```
/
-- Call a common stored procedure.
select autonomous_5(11,22);
-- View the table result.
select * from t2 order by a;
```

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.

```
create table t1(a int ,b text);

START TRANSACTION;
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    dbe_output.print_line('just use call. ');
    insert into t1 values(1,'you are so cute,will commit!');
END;
/
insert into t1 values(1,'you will rollback!');
rollback;

select * from t1;
```

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 main 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:

```
create table t4(a int, b int, c text);

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;
/
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;
/

select autonomous_33(0);

select * from t4;
```

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 1024, 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.
- 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.

- A trigger function does not support autonomous transactions.

```
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

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;
```

- 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).

```
create table t1(a int ,b text);

DECLARE
--PRAGMA AUTONOMOUS_TRANSACTION;
```

```
BEGIN
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  db_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;
/

select * from t1;
```

- Autonomous transactions do not support **ref_cursor** parameter transfer.

```
create table sections(section_ID int);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);

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;
/

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;
/
```

- Distributed autonomous transactions of the IMMUTABLE and STABLE types cannot be pushed down.

```
CREATE OR REPLACE procedure autonomous_test_in_p_116(num1 int )
IMMUTABLE
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
perform pg_sleep(1);
END;
/

CREATE OR REPLACE procedure autonomous_test_in_p_117(num1 int ) STABLE AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
perform pg_sleep(1);
END;
/
```

- The distributed system does not support detection. When a deadlock occurs, a lock waiting timeout error is reported.

```
create table test_lock (id int,a date);
insert into test_lock values (10,sysdate),(11,sysdate),(12,sysdate);
CREATE OR REPLACE FUNCTION autonomous_test_lock(num1 int,num2 int) RETURNS
```

```
integer LANGUAGE plpgsql AS $$  
DECLARE num3 int := 4;  
PRAGMA AUTONOMOUS_TRANSACTION;  
BEGIN  
update test_lock set a=sysdate where id =11;  
RETURN num1+num2+num3;  
END;  
$$;  
start transaction;  
update test_lock set a=sysdate where id =11;  
call autonomous_test_lock(1,1);  
END;
```

- Autonomous transaction functions only return records in the out format.
- The isolation level of an autonomous transaction cannot be changed.
- Autonomous transactions do not support the setof return type.

12 System Catalogs and System Views

12.1 Overview of System Catalogs and System Views

System catalogs store the 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 users with the SYSADMIN permission 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.
 - The `gs_package` system catalog can be queried in a distributed system but is meaningless.
 - 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 users with the SYSADMIN or POLADMIN permission 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 COMMENTS keyword
modifydate	timestamp without time zone	The latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable the policy. • t (true): enabled. • f (false): disabled.

12.2.2 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS records the DML database operations about the unified audit. Only users with the SYSADMIN or POLADMIN permission 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, SELECT , INSERT , and DELETE .
labelname	name	Specifies the resource label name, which corresponds to the polname column in the GS_AUDITING_POLICY system catalog.

Name	Type	Description
policyoid	oid	OID of the audit policy, corresponding to OID in the GS_AUDITING_POLICY 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 users with the SYSADMIN or POLADMIN permission 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 logical_expr .
labelname	name	Name. Currently, the value is logical_expr .
policyoid	oid	OID of the audit policy, corresponding to the OID in the system catalog in GS_AUDITING_POLICY .
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 users with the SYSADMIN or POLADMIN permission 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, CREATE , ALTER , and DROP .
labelname	name	Specifies the resource label name, which corresponds to the polname column in the GS_AUDITING_POLICY system catalog.
policyoid	oid	This column corresponds to the OID in the GS_AUDITING_POLICY 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. • t (true): yes. • f (false): no.
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 cn_id is in the key of the unique query.
unique_query	text	Standardized UniqueSQL text string.
locktag	text	Information of a lock that the session waits for. It can be parsed using locktag_decode .
lockmode	text	Mode of a lock that the session waits for. • LW_EXCLUSIVE: exclusive lock. • LW_SHARED: shared lock. • LW_WAIT_UNTIL_FREE: waits for the LW_EXCLUSIVE to be available.

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 value can be active , idle in transaction , fastpath function call , idle in transaction (aborted) , disabled , or retrying .

12.2.6 GS_DB_PRIVILEGE

GS_DB_PRIVILEGE records the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 12-6 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 Table 7-114 .
admin_option	boolean	Whether the ANY permission recorded in the privilege_type column can be re-granted • t: yes • f: no

12.2.7 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**.

Table 12-7 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 name, or parameter value required by users during cluster initialization.

12.2.8 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.

Table 12-8 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 usernames.
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.

12.2.9 GS_JOB_ARGUMENT

GS_JOB_ARGUMENT provides the parameter attributes of DBE_SCHEDULER scheduled tasks and programs.

Table 12-9 GS_JOB_ARGUMENT columns

Name	Type	Description
oid	oid	Row identifier (hidden column)

Name	Type	Description
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.

12.2.10 GS_MASKING_POLICY

GS_MASKING_POLICY records the main information about dynamic data masking policies. Each record corresponds to a masking policy. Only users with the SYSADMIN or POLADMIN permission can access this system catalog.

Table 12-10 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 COMMENTS keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified.
polenabled	boolean	Specifies whether to enable the policy. • t (true): enabled • f (false): disabled

12.2.11 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 users with the SYSADMIN or POLADMIN permission can access this system catalog.

Table 12-11 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 specified in 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 GS_MASKING_POLICY .
actmodifydate	timestamp without time zone	Latest timestamp when a record is created or modified

12.2.12 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 users with the SYSADMIN or POLADMIN permission can access this system catalog.

Table 12-12 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 logical_expr .
filterlabelname	name	Filtering range. Currently, the value is logical_expr .
policyoid	oid	OID of a masking policy to which a record belongs, corresponding to the OID in GS_MASKING_POLICY .

Name	Type	Description
modifydate	timestamp without time zone	Latest timestamp when a user filter criterion is created or modified.
logicaloperator	text	Polish notation of the filter criteria.

12.2.13 GS_MATVIEW

GS_MATVIEW provides information about each materialized view in the database.

Table 12-13 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 complete-refresh materialized view does not correspond to a map table, the value of this column is 0 .
ivm	boolean	Type of a materialized view. The value t indicates a fast-refresh materialized view, and the value f indicates a complete-refresh materialized view.
needrefresh	boolean	Reserved column.
refreshtime	timestamp	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.

12.2.14 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 12-14 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	int4	Reserved column.

12.2.15 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.

12.2.16 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 users with the SYSADMIN or POLADMIN permission can access this system catalog.

Fully Qualified Domain Name (FQDN) identifies an absolute path of a database resource.

Table 12-15 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 RESOURCE .

Name	Type	Description
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.
fqdtype	name	Type of the identified database resource, for example, schema, table, column, or view.

12.2.17 GS_RECYCLEBIN

GS_RECYCLEBIN describes details about objects in the recycle bin of the flashback feature.

Currently, the distributed system does not support the flashback feature.

12.2.18 GS_SQL_PATCH

GS_SQL_PATCH stores the status information about all **SQL_PATCH** statements. Currently, this function is not supported in distributed mode.

Table 12-16 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.

Name	Type	Description
original_query	text	Original statement (reserved column).
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.

12.2.19 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.

Currently, the distributed system does not support the flashback feature.

12.2.20 GS_UID

GS_UID records the unique identification meta information of the **hasuids** attribute table in the database.

Table 12-17 GS_UID columns

Name	Type	Description
relid	oid	OID of a table
uid_backup	bigint	Largest unique identifier that can be assigned to a table

12.2.21 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 12-18 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	prname in PG_PROC	PG_PROC prname of the aggregate function
aggtransfn	regproc	prname in PG_PROC	Transition function
aggcollectfn	regproc	prname in PG_PROC	Collect function
aggfinalfn	regproc	prname in PG_PROC	Final function (0 if none)
aggsortop	oid	oid in PG_OPERATOR	Associated sort operator (0 if none)
aggtranstype	oid	oid in PG_TYPE	Data type of the aggregate function's internal transition (state) data The possible values and their meanings are defined by the types in pg_type.h . 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.
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: • n: normal aggregate • o: ordered set aggregate

Name	Type	Reference	Description
aggrnumdirect 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 0 .

12.2.22 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 12-19 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 (0 if the access method does not have a fixed set of operator strategies).
amsupport	smallint	-	Number of support routines for the access method.
amcanorder	boolean	-	Specifies whether the access method supports ordered scans sorted by the indexed column's value. • t (true): supported. • f (false): not supported.
amcanorderbyop	boolean	-	Specifies whether the access method supports ordered scans sorted by the result of an operator on the indexed column. t (true): supported. f (false): not supported.

Name	Type	Reference	Description
amcanbackward	boolean	-	Specifies whether the access method supports backward scanning. • t (true): supported. • f (false): not supported.
amcanunique	boolean	-	Specifies whether the access method supports unique indexes. • t (true): supported. • f (false): not supported.
amcanmulticol	boolean	-	Specifies whether the access method supports multi-column indexes. • t (true): supported. • f (false): not supported.
amoptionalkey	boolean	-	Specifies whether the access method supports scanning without any constraint for the first index column. • t (true): supported. • f (false): not supported.
amsearcharray	boolean	-	Specifies whether the access method supports ScalarArrayOpExpr searches. t (true): supported. f (false): not supported.
amsearchnulls	boolean	-	Specifies whether the access method supports IS NULL/NOT NULL searches. • t (true): supported. • f (false): not supported.
amstorage	boolean	-	Specifies whether the index storage data type can differ from the column data type. • t (true): allowed. • f (false): not allowed.
amclusterable	boolean	-	Specifies whether an index of this type can be clustered on. • t (true): allowed. • f (false): not allowed.

Name	Type	Reference	Description
ampredlocks	boolean	-	Specifies whether an index of this type manages fine-grained predicate locks. • t (true): allowed. • f (false): not allowed.
amkeytype	oid	oid in PG_TYPE	Type of data stored in index (0 if it is not a fixed type)
aminsert	regproc	prname in PG_PROC	"Insert this tuple" function
ambeginscan	regproc	prname in PG_PROC	"Prepare for index scan" function
amgettupl e	regproc	prname in PG_PROC	"Next valid tuple" function (0 if none)
amgetbitm ap	regproc	prname in PG_PROC	"Fetch all valid tuples" function (0 if none)
amrescan	regproc	prname in PG_PROC	"(Re)start index scan" function
amendsca n	regproc	prname in PG_PROC	"Clean up after index scan" function
ammarkpos	regproc	prname in PG_PROC	"Mark current scan position" function
amrestrpos	regproc	prname in PG_PROC	"Restore marked scan position" function
ammerge	regproc	prname in PG_PROC	"Merge multiple indexes" function
ambuild	regproc	prname in PG_PROC	"Build new index" function
ambuilde mpty	regproc	prname in PG_PROC	"Build empty index" function
ambulkdel ete	regproc	prname in PG_PROC	Bulk-delete function
amvacuum cleanup	regproc	prname in PG_PROC	Post-VACUUM cleanup function
amcanretu rn	regproc	prname in PG_PROC	Function to check whether the index supports index-only scans (0 if none)
amcostesti mate	regproc	prname in PG_PROC	Function to estimate cost of an index scan

Name	Type	Reference	Description
amoptions	regproc	prname in PG_PROC	Function to parse and validate reloptions for an index

12.2.23 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 12-20 PG_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
amopfamily	oid	oid in PG_OPFAMILY	Operator family of this entry.
amoplefttype	oid	oid in PG_TYPE	Left-hand input data type of the operator For details about the possible values and their description, see Data Type .
amoprightrighttype	oid	oid in PG_TYPE	Right-hand input data type of the operator For details about the possible values and their description, see Data Type .
amopstrategy	smallint	-	Number of operator strategies.
amoppurpose	"char"	-	Purpose of the operator. • s: search • o: order
amopopr	oid	oid in PG_OPERATOR	OID of the operator.
amopmethod	oid	oid in PG_AM	Operator family of the index access method.

Name	Type	Reference	Description
amopsortfamily	oid	oid in PG_OPFAMILY	The B-tree operator family according to which this entry sorts for an ordering operator (0 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.

12.2.24 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 12-21 PG_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amprocfamily	oid	oid in PG_OPFAMILY	Operator family of this entry
amproclefttype	oid	oid in PG_TYPE	Left-hand input data type of the associated operator For details about common data types, see Data Type .
amprocrighttype	oid	oid in PG_TYPE	Right-hand input data type of the associated operator For details about common data types, see Data Type .
amprocnum	smallint	-	Support procedure number
amproc	regproc	PG_PROC .prcname	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**.

12.2.25 PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING provides load mapping group information in the database.

Table 12-22 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

12.2.26 PG_ATTRDEF

PG_ATTRDEF records default values of columns.

Table 12-23 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 s indicates that the column is a generated column, and the value '\0' indicates that the column is an ordinary column. The default value is '\0' .

12.2.27 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 12-24 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 ANALYZE. • The value 0 indicates that no statistics should be collected. • A negative value indicates that the system default statistic object is used. • The exact meaning of positive values is data type-dependent. For scalar data types, attstattarget is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	smallint	Copy of PG_TYPE in the column's type.
attnum	smallint	Number of the column
atndims	integer	Number of dimensions if the column is an array (0 in other cases)
attcacheoff	integer	This column is always -1 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 varchar 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 -1 for types that do not need ATTTYPMOD.
attbyval	boolean	Copy of PG_TYPE of this column's type.
attstorage	"char"	Copy of PG_TYPE of this column's type.
attalign	"char"	Copy of PG_TYPE of this column's type.
attnotnull	boolean	A NOT NULL constraint. It is possible to change this column to enable or disable the constraint.

Name	Type	Description
attahasdef	boolean	This column has a default value, in which case there will be a corresponding entry in the PG_ATTRDEF table 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.
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: • n_distinct: number of distinct values of a column (excluding subtables). • n_distinct_inherited: number of distinct values of a column (including subtables).
attfdwoptions	text[]	Column attribute of a foreign table. Currently, dist_fdw , file_fdw , and log_fdw do not use foreign table column attributes.
attinitdefval	bytea	attinitdefval stores the default value expression. ADD COLUMN in the row-store table must use this column.
attkvtype	tinyint	Specifies a key value type for a column. Types include: 0. ATT_KV_UNDEFINED: default value. 1. ATT_KV_TAG: dimension. 2. ATT_KV_FIELD: indicator. 3. ATT_KV_TIMETAG: time column. 4. ATT_KV_HIDETAG: hidden distribution key.

12.2.28 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 users with the SYSADMIN permission.

Table 12-25 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 • t (true): yes • f (false): no
rolinherit	boolean	Whether the role automatically inherits permissions of roles of which it is a member • t (true): automatically inherited • f (false): not automatically inherited
rolcreaterole	boolean	Whether the role can create more roles • t (true): yes • f (false): no
rolcreatedb	boolean	Whether the role can create databases • t (true): yes • f (false): no
rolcatupdate	boolean	Whether the role can directly update system catalogs Only the initial system administrator whose usesysid is 10 has this permission. It is unavailable for other users. • t (true): yes • f (false): no
rolcanlogin	boolean	Whether the role can log in (whether this role can be given as the initial session authorization identifier) • t (true): yes • f (false): no

Name	Type	Description
rolreplication	boolean	Specifies whether the role has the replication permission. • t (true): yes • f (false): no
rolauditadmin	boolean	Specifies whether the role has the AUDITADMIN permission. • t (true): yes • f (false): no
rolsystemadmin	boolean	Specifies whether the role has the SYSADMIN permission. • t (true): yes • f (false): no
rolconnlimit	integer	Maximum number of concurrent connections that this role can make (valid for roles that can log in) The value -1 indicates there is no limit.
rolpassword	text	Password ciphertext (NULL if no password).
rolvalidbegin	timestamp with time zone	Account validity start time (NULL if no start time)
rolvaliduntil	timestamp with time zone	Password expiry time (NULL if no expiration)
roluseft	boolean	Whether the role can perform operations on foreign tables • t (true): yes • f (false): no
rolparentid	oid	OID of a group user to which the user belongs
roltabspace	text	Maximum size of a user data table
rolkind	"char"	Special user types, including node group administrators, permanent users, and common users.
roltemp space	text	Maximum size of a user's temporary table, in KB.
rolspillspace	text	Maximum size of data that can be written to disks when a user executes a job, in KB.
rolexcpdata	text	Query rules that can be set by users (reserved)

Name	Type	Description
rolmonitoradmin	boolean	Specifies whether the role has the MONADMIN permission. • t (true): yes • f (false): no
roloperatoradmin	boolean	Specifies whether the role has the O&M administrator permission. • t (true): yes • f (false): no
rolpolicyadmin	boolean	Specifies whether the role has the POLADMIN permission. • t (true): yes • f (false): no

12.2.29 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of a role. This system catalog is accessible only to users with the SYSADMIN permission.

Table 12-26 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 password_encryption_type .

12.2.30 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership between roles.

Table 12-27 PG_AUTH_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member.

Name	Type	Description
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 true (yes) and false (no).

12.2.31 PG_CAST

PG_CAST records the conversion relationship between data types.

Table 12-28 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 (0 if no conversion function is required)
castcontext	"char"	Conversion mode between the source and target data types. • e: Only explicit conversion can be performed (using the CAST or :: syntax). • i: Implicit conversion can be performed. • a: Both explicit and implicit conversion can be performed between data types.
castmethod	"char"	Conversion method. • f: Conversion is performed using the specified function in the castfunc column. • b: Binary forcible conversion rather than the specified function in the castfunc column is performed between data types.

12.2.32 PG_CLASS

PG_CLASS records database objects and their relationship.

Table 12-29 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 PG_TYPE records.
reloftype	oid	OID of the composite type (0 for other types).
relowner	oid	Owner of the relationship.
relam	oid	Access method used, such as B-tree and hash, if this is an index.
relfilenode	oid	Name of the on-disk file of this relationship (0 if such file does not exist).
reltablespace	oid	Tablespace in which this relationship is stored. If the value is 0 , 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 VACUUM , ANALYZE , and a few DDL statements such as CREATE INDEX .
reltoastrelid	oid	OID of the TOAST table associated with the table (0 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 (0 for a table other than a TOAST table).

Name	Type	Description
relhasindex	boolean	Its value is true 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 relhasindex column and sets the value to false .
relisshared	boolean	Its value is true if the table is shared across all databases in the entire cluster. Otherwise, the value is false . Only certain system catalogs (such as PG_DATABASE) are shared.
relpersistence	"char"	• p: permanent table. • u: non-log table. • t: temporary table. • g: global temporary table.
relkind	"char"	• r: ordinary table. • i: index. • s: sequence. • v: view. • c: composite type. • t: TOAST table. • f: foreign table. • m: materialized view. • e: STREAM object. • o: CONTVIEW object.
relnatts	smallint	Number of user columns in the relationship (excluding system columns). PG_ATTRIBUTE 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 PG_CONSTRAINT .
relhasoids	boolean	Its value is true if an OID is generated for each row of the relationship. Otherwise, the value is false .
relhaspkey	boolean	Its value is true if the table has (or once had) a primary key. Otherwise, the value is false .
relhasrules	boolean	Its value is true if the table has rules. For details, see the system catalog PG_REWRITE .
relhastriggers	boolean	The value is true if the table has (or once had) triggers. Triggers of the table and view are recorded in the system catalog PG_TRIGGER .

Name	Type	Description
relhassubclass	boolean	Its value is true if the table has (or once had) any inheritance child table. Otherwise, the value is false .
relrowmovement	boolean	Specifies whether row migration is allowed when the partitioned table is updated. • true: Row migration is allowed. • false: Row migration is not allowed.
parttype	"char"	Specifies whether the table or index has the property of a partitioned table. • p: The table or index has the property of a partitioned table. • n: The table or index does not have the property of a partitioned table.
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 0 (InvalidTransactionId) if the relationship is not a table. To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information.
relacl	aclitem[]	Access permissions. The command output of the query is as follows: rolename=xxxx/yyyy -- Assigning permissions to a role =xxxx/yyyy -- Assigning the permission to public xxxx indicates assigned permissions, and yyyy indicates roles with the assigned permissions. For details on permission descriptions, see Table 12-30 .
reloptions	text[]	Table or index access method, using character strings in the format of <i>keyword=value</i> .
relreplident	"char"	Identifier of a decoding column in logical decoding. • d: default (primary key, if any). • n: none. • f: all columns. • i: The indisreplident of the index is specified or the default index is used.

Name	Type	Description
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 0 (InvalidTransactionId) 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 PG_HASHBUCKET catalog. NULL indicates that hash bucket shards are not included.
relbucketkey	int2vector	Hash partition key information. NULL 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 pg_clog to be shrunk. The value is 0 (InvalidTransactionId) if the relationship is not a table.

Table 12-30 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

Parameter	Parameters
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	Authorization options for preceding permissions

12.2.33 PG_COLLATION

PG_COLLATION describes available collations, which are essentially mappings from an SQL name to OS local categories.

Table 12-31 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)
collnamespace	oid	oid in PG_NAMESPACE	OID of the namespace that contains this collation
collowner	oid	oid in PG_AUTHID	Owner of the collation
collencoding	integer	-	Encoding in which the collation is applicable, or -1 if it works for any encoding. It is compatible with PostgreSQL.
collcollate	name	-	LC_COLLATE for this collation object
collctype	name	-	LC_CTYPE for this collation object

12.2.34 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, unique, and foreign key constraints on tables.

Table 12-32 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.
contype	"char"	• c: check constraint • p: primary key constraint • u: unique constraint • t: trigger constraint • x: mutual exclusion constraint • f: foreign key constraint • s: clustering constraint • i: invalid constraint
condeferrable	boolean	Specifies whether the constraint is deferrable. • true: yes • false: no
condeferred	boolean	Specifies whether the constraint can be deferrable by default. • true: yes • false: no
convalidated	boolean	Specifies whether the constraint is valid. Currently, it can be set to false only for foreign key and check constraints. • true: valid • false: invalid
conrelid	oid	Table containing this constraint (0 if it is not a table constraint).
contypid	oid	Domain containing this constraint (0 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 0 .

Name	Type	Description
confupdtype	"char"	Foreign key update action code. • a: no action • r: restriction. • c: cascading. • n: The parameter is set to null. • d: The default value is used.
confdeltype	"char"	Foreign key deletion action code. • a: no action • r: restriction. • c: cascading. • n: The parameter is set to null. • d: The default value is used.
confmatchtype	"char"	Foreign key match type. • f: full match • p: partial match • u: unspecified (The NULL value can be matched if f is specified.)
conislocal	boolean	Specifies whether the constraint is defined locally for the relation. • true: yes • false: no
coninhcount	integer	Number of direct inheritance parent tables that this constraint has. When the value is not 0, the constraint cannot be deleted or renamed.
connoinherit	boolean	Specifies whether the constraint can be inherited. • true: yes • false: no
consoft	boolean	Specifies whether the column indicates an informational constraint. • true: yes • false: no
conopt	boolean	Specifies whether you can use the informational constraint to optimize the execution plan. • true: yes • false: no

Name	Type	Description
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.
conpfeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key.
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 INDEX .

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 **PG_CLASS** must agree with the number of check-constraint entries found in the table for each relationship.

12.2.35 PG_CONVERSION

PG_CONVERSION describes encoding conversion information.

Table 12-33 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)

Name	Type	Reference	Description
connamespace	oid	oid in PG_NAMESPACE	OID of the namespace that contains this conversion
conowner	oid	oid in PG_AUTHID	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID
conproc	regproc	prname in PG_PROC	Conversion procedure
condefault	boolean	-	If this is the default conversion, the value is true . Otherwise, the value is false .

12.2.36 PG_DATABASE

PG_DATABASE records information about available databases.

Table 12-34 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. • true: allowed. • false: not allowed.
datallowconn	boolean	This column is used to protect the template0 database from being altered. • true: Users can connect to the database. • false: No user can connect to the database.

Name	Type	Description
datconlimit	integer	Maximum number of concurrent connections allowed on this database. The value -1 indicates no limit.
datlastsysoid	oid	Last system OID in the database.
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 datfrozenxid64 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 relminmxid in PG_CLASS of all tables in the database.

12.2.37 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 12-35 PG_DB_ROLE_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items (0 if no database is specified).
setrole	oid	Role corresponding to the configuration items (0 if no role is specified).

Name	Type	Description
setconfig	text[]	Default value of GUC items.

12.2.38 PG_DEFAULT_ACL

PG_DEFAULT_ACL records initial permissions assigned to newly created objects.

Table 12-36 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 (0 if no ID).
defaclobjtype	"char"	Object type of the permission. • r: table or view • S: sequence • f: function • T: type • K: client master key • k: column encryption key
defaclacl	aclitem[]	Access permissions that this type of object should have on creation.

12.2.39 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 12-37 PG_DEPEND columns

Name	Type	Reference	Description
classid	oid	oid in PG_CLASS	OID of the system catalog where a dependent object resides

Name	Type	Reference	Description
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	Column number for a table column (objid and classid refer to the table itself); 0 for all other object types
refclassid	oid	oid in PG_CLASS	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 (refobjid and refclassid refer to the table itself); 0 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**.
- **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.

12.2.40 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 an entire cluster.

Table 12-38 PG_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this description pertains to
classoid	oid	oid in PG_CLASS	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a comment on a table column (objoid and classoid refer to the table itself); 0 for all other object types
description	text	-	Arbitrary text that serves as the description of the object

12.2.41 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 permissions of the built-in role **gs_role_directory_create** can create directory objects.

Table 12-39 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.

12.2.42 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 12-40 PG_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
enumtypid	oid	oid in PG_TYPE	OID of the PG_TYPE entry 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.

12.2.43 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 12-41 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	oid in PG_AUTHID	Owner of the foreign server

Name	Type	Reference	Description
srvfdw	oid	oid in PG_FOREIGN_DATA_WRAPPER	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

12.2.44 PG_HASHBUCKET

PG_HASHBUCKET records hash bucket information.

Table 12-42 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 DN.
bucketref	integer	Reserved column with 1 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 PG_HASHBUCKET data in the same row.

12.2.45 PG_INDEX

PG_INDEX records part of index information. The rest is mostly recorded in **PG_CLASS**.

Table 12-43 PG_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the PG_CLASS entry for the index.
indrelid	oid	OID of the PG_CLASS entry for the table that uses the index.
indnatts	smallint	Number of columns in the index
indisunique	Boolean	Specifies whether the index is unique. • true: The index is unique. • false: The index is not unique.
indisprimary	Boolean	Specifies whether the index is the primary key of the table. • true: The index is the primary key of the table. indisunique should always be true when the value of this column is true. • false: The index is not the primary key of the table.
indisexclusion	Boolean	Specifies whether the index supports exclusive constraints. • true: The index supports exclusive constraints. • false: The index does not support exclusive constraints.
indimmediate	Boolean	Specifies whether to check the uniqueness of data to be inserted. • true: The uniqueness check is performed immediately when data is inserted. • false: The uniqueness check is not performed when data is inserted.
indisclustered	Boolean	Specifies whether the table is clustered on the index. • true: The table is clustered on the index. • false: The table is not clustered on the index.
indisusable	Boolean	Specifies whether the index is available for INSERT and SELECT operations. • true: The index is available for INSERT and SELECT operations. • false: The index is unavailable for INSERT and SELECT operations.

Name	Type	Description
indisvalid	Boolean	• true: The index can be used for query. • false: 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 true.
indcheckxmin	Boolean	• true: Queries must not use the index until the xmin of this row in PG_INDEX is lower than their TransactionXmin, because the table may contain broken HOT chains with incompatible rows that they can see. • false: Indexes can be used for query.
indisready	Boolean	• true: The index is available for inserting data. • false: The index is ignored when data is inserted or modified.
indkey	int2vector	This is an array of indnatts values indicating that this index creates table columns. For example, a value of 1 3 indicates that the first and the third columns make up the index key. The value 0 in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference.
indcollation	oidvector	ID of each column used by the index.
indclass	oidvector	For each column in the index key, this contains the OID of the operator class to use. See PG_OPCLASS 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.

Name	Type	Description
indisreplident	Boolean	Specifies whether the column of this index is a decoded column of logical decoding. • true: The column of this index is a decoded column of logical decoding. • false: The column of this index is not a decoded column of logical decoding.
indnkeyatts	smallint	Total number of columns in the index. The columns that exceed the value of indnatts are not involved in the index query.

12.2.46 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 12-44 PG_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	oid in PG_CLASS	OID of a child table
inhparent	oid	oid in PG_CLASS	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.

12.2.47 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.

Table 12-45 PG_JOB columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).

Name	Type	Description
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 -1 , 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.
job_status	"char"	Execution status of the current task. The default value is 's'. The options are as follows: • 'r': running • 's': successfully finished • 'f': job failed • 'd': disable If a job fails to be executed for 16 consecutive times, job_status is automatically set to 'd', and no more attempt will be made on this job. Note: When you disable a scheduled task (by setting job_queue_processes 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 (job_queue_processes is not 0), the system updates the value of this column based on the real-time job status.
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.

Name	Type	Description
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.
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: • true: enabled. • false: disabled.
failure_message	text	Error information about the latest task execution.

12.2.48 PG_JOB_PROC

PG_JOB_PROC records the content of each job in the [PG_JOB](#) table, 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.

Table 12-46 PG_JOB_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
job_id	integer	Foreign key, which is joined with job_id in the system catalog PG_JOB .
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.

12.2.49 PG_LANGUAGE

PG_LANGUAGE registers programming languages. You can use them and APIs to write functions or stored procedures.

Table 12-47 PG_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
lanname	name	-	Language name.
lanowner	oid	oid in PG_AUTHID	Owner of the language.
lanispl	boolean	-	• true: user-defined language. • false: internal language, for example, SQL. 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	-	• true: The language is trusted, which means that it is believed not to grant access to anything outside the normal SQL execution environment. • false: The language is untrusted. Only the initial user can create functions in untrusted languages.
lanplcallfoid	oid	oid in PG_PROC	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	oid in PG_PROC	This column references a function responsible for executing "inline" anonymous code blocks (DO blocks). The value is 0 if inline blocks are not supported.

Name	Type	Reference	Description
lanvalidator	oid	oid in PG_PROC	This column references a language validator function responsible for checking the syntax and validity of new functions when they are created. The value is 0 if no validator is provided.
lanacl	aclitem[]	-	Access permission.

12.2.50 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 users with the SYSADMIN permission.

Table 12-48 PG_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	oid in PG_LARGEOBJECT_METADATA	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 LOBLKSIZE 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 are read as zeroes.

12.2.51 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 12-49 PG_LARGEOBJECT_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lomowner	oid	oid in PG_AUTHID	Owner of the large object
lomacl	aclitem[]	-	Access permissions

12.2.52 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 12-50 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.
nsacl	aclitem[]	Access permission. For details, see GRANT and REVOKE .
in_redistribution	"char"	Specifies whether the content is in the redistribution state. • t (true): yes • f (false): no
nspblockchain	Boolean	• If the value is true, the tamper-proof mode is used. • If the value is false, the non-tamper-proof mode is used.

12.2.53 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 12-51 PG_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier.
object_type	"char"	Object type. • r: ordinary table • i: index • s: sequence • v: view • p: stored procedure and function.
creator	oid	ID of a creator.
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 ALTER , GRANT , and REVOKE .
createcsn	bigint	CSN when an object is created.
changeccsn	bigint	CSN when DDL operations are performed on a table or an index.

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 **changeccsn**) of the transaction to which the table or index belongs is recorded. Because the creation time of the object cannot be obtained, **ctime** and **createccsn** 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.
- **createccsn** and **changeccsn** record the transaction commit sequence number of the transaction to which the current operation belongs.

12.2.54 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 12-52 PG_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opcmethod	oid	oid in PG_AM	Index access method operator class served by an operator class
opcname	name	-	Name of the operator class
opcnamespace	oid	OID in PG_NAMESPACE	Namespace of the operator class
opcowner	oid	oid in PG_AUTHID	Owner of the operator class
opcfamily	oid	oid in PG_OPFAMILY	Operator family containing the operator class
opcintype	oid	oid in PG_TYPE	Data type that the operator class indexes
opcdefault	boolean	-	The value is true if this operator class is the default for opcintype .
opckeytype	oid	oid in PG_TYPE	Type of data stored in index, or zero if same as opcintype

An operator class's **opcmethod** must match the **opfmeth** of its containing operator family.

12.2.55 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 12-53 PG_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)

Name	Type	Reference	Description
oprname	name	-	Name of an operator
oprnamespace	oid	oid in PG_NAMESPACE	OID of the namespace that contains the operator.
oprowner	oid	oid in PG_AUTHID	Owner of the operator.
oprkind	"char"	-	• b: infix ("both"). • l: prefix ("left"). • r: postfix ("right").
oprcanmerge	boolean	-	Whether the operator supports merge joins • t (true): yes • f (false): no
oprcanhash	boolean	-	Whether the operator supports hash joins • t (true): yes • f (false): no
oprleft	oid	oid in PG_TYPE	Type of the left operand.
oprright	oid	oid in PG_TYPE	Type of the right operand.
oprresult	oid	oid in PG_TYPE	Type of the result.
oprcom	oid	oid in PG_OPERATOR	If it exists, the value is the exchange character of this operator. If it does not exist, the value is 0 .
oprnegate	oid	oid in PG_OPERATOR	If it exists, the value is the inverter of this operator. If it does not exist, the value is 0 .
oprcode	regproc	proname in PG_PROC	Function that implements the operator.
oprrest	regproc	proname in PG_PROC	Restriction selectivity estimation function for the operator.
oprjoin	regproc	proname in PG_PROC	Join selectivity estimation function for the operator.

12.2.56 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 12-54 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opfmethod	oid	PG_AM.oid	Index access method used by an operator family
opfname	name	-	Name of the operator family
opfnamespace	oid	oid in PG_NAMESPACE	Namespace of the operator family
opfowner	oid	oid in PG_AUTHID	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](#).

12.2.57 PG_PARTITION

PG_PARTITION records all partitioned tables, table partitions, TOAST tables on table partitions, and index partitions in the database. Partitioned index information is not stored in the system catalog PG_PARTITION.

Table 12-55 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. • r: partitioned table • p: table partition • x: index partition • t: TOAST table.

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. • r: range partition • v: numeric partition.
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. • t (true): yes • f (false): no
reldeltarelid	oid	OID of a Delta table.
reldeltaidx	oid	OID of the index for a Delta table.
relcudescrelid	oid	OID of a CU description table.
relcudescidx	oid	OID of the index for a CU description table.

Name	Type	Description
relfrozenxid	xid32	Frozen transaction ID. To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information.
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 pg_class.reloptions , it is expressed in a string in the format of keyword=value.
relfrozenxid64	xid	Frozen transaction ID.
relminmxid	xid	Frozen multi-transaction ID.

12.2.58 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 12-56 PG_PLTEMPLATE columns

Name	Type	Description
tmplname	name	Name of the language for which this template is used
tmpltrusted	boolean	The value is true if the language is considered trusted. Otherwise, the value is false .
tmpldbcreate	boolean	The value is true if the language is created by the owner of the database. Otherwise, the value is false .
tmplhandler	text	Name of the call handler function

Name	Type	Description
tmpline	text	Name of the anonymous block handler (NULL if no name of the block handler exists)
tmplvalidator	text	Name of the verification function (NULL if no verification function is available)
tmpliblibrary	text	Path of the shared library that implements languages
tmplacl	aclitem[]	Access permissions for template (not yet used)

12.2.59 PG_PROC

PG_PROC records information about functions or procedures.

Table 12-57 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.
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. • t (true): yes • f (false): no
proiswindow	Boolean	Specifies whether the function is a window function. • t (true): yes • f (false): no

Name	Type	Description
prosecdef	Boolean	Specifies whether the function is a security definer (or a setuid function). • t (true): yes • f (false): no
proleakproof	Boolean	Specifies whether the function has side effects. If no leakproof treatment is provided for parameters, the function throws errors. • t (true): There is no side effect. • f (false): There are side effects.
proisstrict	Boolean	• t (true): When this function is used, the function does not call the function and returns null if the input parameter is null. • f (false): When this function is used, it is called even if the input parameter is null. Therefore, this function must process null input.
proretset	Boolean	Specifies whether the return value of the function is a set (that is, multiple values of the specified data type). • true: The return value of the function is a set. • false: The return value of the function is not a set.
provolatile	"char"	Specifies whether the function's result depends only on its input parameters, or is affected by outside factors. • i: for "immutable" functions, which always deliver the same result for the same inputs. • s: stable functions, whose results (for fixed inputs) do not change within a scan. • v: volatile functions, whose results may change at any time. It is also used for functions with side-effects, so that the engine cannot get optimized if volatile functions are called.
pronargs	smallint	Number of parameters.
pronargdefaults	smallint	Number of parameters that have default values.
prorettype	oid	Data type of return values.

Name	Type	Description
proargtypes	oidvector	Array that stores the data types of function parameters. This array includes only input parameters (including INOUT 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 OUT and INOUT parameters); however, if all the parameters are IN parameters, this column is null. Note that array indexing is 1-based, whereas for historical reasons, proargtypes is indexed from 0.
proargmodes	"char"[]	Array with the modes of the function parameters, encoded as follows: • i indicates the IN parameter. • o indicates the OUT parameter. • b indicates the INOUT parameter. • v indicates the VARIADIC parameter. If all the parameters are IN parameters, this column is null. Note that indexes correspond to positions of proallargtypes , not proargtypes .
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 proallargtypes , not proargtypes .
proargdefaults	pg_node_tree	Expression tree of the default value. This is the list of pronargdefaults 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 GRANT and REVOKE .

Name	Type	Description
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. • true: fence mode. In this mode, the function is executed in the fork process. • false: not fence mode. For built-in functions, fencedmode is set to false, indicating the not fence mode.
proshippable	Boolean	Specifies whether the function can be pushed down to DNs for execution. The default value is false. • 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.
propackage	Boolean	Specifies whether the function supports overloading. The default value is false. • t (true): supported. • f (false): not supported.
prokind	"char"	Specifies whether the object is a function or a stored procedure. • 'f': The object is a function. • 'p': The object is a stored procedure.
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 NULL .
proisprivate	Boolean	Specifies whether a function is a private function in the package. The default value is false .
propackageid	oid	OID of the package to which the function belongs. If the function is not in the package, the value is 0 .

Name	Type	Description
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 INOUT parameters), and indicates the call signature (API) of the function.
prodefaultargposext	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 INOUT 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 INOUT parameters).

12.2.60 PG_RANGE

PG_RANGE records information about range types, except for records of types in [PG_TYPE](#).

Table 12-58 PG_RANGE columns

Name	Type	Reference	Description
rngtypeid	oid	oid in PG_TYPE	OID of the range type
rngsubtype	oid	oid in PG_TYPE	OID of the element type (subtype) of this range type
rngcollation	oid	oid in PG_COLLATION	OID of the collation used for range comparisons (0 if none)
rngsubopc	oid	oid in PG_OPCLASS	OID of the subtype's operator class used for range comparisons
rngcanonical	regproc	prname in PG_PROC	Name of the function to convert a range value into canonical form (0 if none)

Name	Type	Reference	Description
rngsubdiff	regproc	prname in PG_PROC	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 rngsubdiff is 0 .

If the element type is discrete, **rngcanonical** determines the collation used for the range type. If the element type is not collatable, **rngsubopc** determines the collation used for the range type. If the element type is collatable, **rngsubopc** and **rngcollation** determine the collation used for the range type.

12.2.61 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 12-59 PG_REPLICATION_ORIGIN columns

Name	Type	Description
roident	oid	Unique replication source identifier within a cluster.
roname	text	External user-defined replication source name.

12.2.62 PG_RESOURCE_POOL

PG_RESOURCE_POOL provides information about database resource pools.

Table 12-60 PG_RESOURCE_POOL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
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.

Name	Type	Description
control_group	name	Name of the Cgroup where the resource pool is located.
active_statements	integer	Maximum number of concurrent statements in the resource pool
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 10,000 per second for row store.
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%.
nodegroup	name	Indicates the name of the node group to which the resource pool belongs.
is_foreign	Boolean	Specifies whether the resource pool can be used for users outside the node group. If it is set to true , the resource pool controls the resources of common users who do not belong to the current resource pool. If it is set to false , the resources of common users who do not belong to the current resource pool are not controlled.
max_worker	integer	Concurrency in a table during data redistribution. This column is used only for scaling.

12.2.63 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 12-61 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 0 to indicate the entire table).

Name	Type	Description
ev_type	"char"	Event type for the rule. • 1: SELECT • 2: UPDATE • 3: INSERT • 4: DELETE
ev_enabled	"char"	Controls the mode in which the rule is triggered. • O: The rule is triggered in origin and local modes. • D: The rule is disabled. • R: The rule is triggered in replica mode. • A: The rule always fires.
is_instead	Boolean	The value is true if the rule is of the INSTEAD type. Otherwise, the value is false .
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

12.2.64 PG_RLSPOLICY

PG_RLSPOLICY records row-level security policies.

Table 12-62 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. • t: OR condition concatenation of expressions. • f: AND condition concatenation of expressions.

Name	Type	Description
polroles	oid[]	OID list of users affected by the row-level security policy. If this parameter is not specified, all users are affected.
polqual	pg_node_tree	Expression of a row-level security policy.

12.2.65 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 12-63 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	oid in PG_CLASS	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

12.2.66 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 a cluster. That is, there is only one **PG_SHDEPEND** for each cluster, not for each database.

Table 12-64 PG_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	oid in PG_DATABASE	OID of the database the dependent object is in, or zero for a shared object.
classid	oid	oid in PG_CLASS	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 (objid and classid refer to the table itself); 0 for all other object types
refclassid	oid	oid in PG_CLASS	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 below.

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 [GS_DB_PRIVILEGE](#) system catalog).

12.2.67 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 a cluster. That is, there is only one PG_SHDESCRIPTION for each cluster, not for each database.

Table 12-65 PG_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the described object.
classoid	oid	oid in PG_CLASS	OID of the system catalog where the object appears.
description	text	-	Description of the object.

12.2.68 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 in a cluster. There is only one copy of PG_SHSECLABEL in a database cluster, not one per database.

Table 12-66 PG_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	oid in PG_CLASS	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

12.2.69 PG_STATISTIC

PG_STATISTIC records statistics about tables and index columns in a database. By default, only users with the SYSADMIN permission can access the system catalog. Common users can access the system catalog only after being authorized.

Table 12-67 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. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the distinct value to the total number of rows. For example, if stadistinct is -0.5, the actual distinct value is the total number of rows multiplied by 0.5. • The value 0 indicates that the number of distinct values is unknown.
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 NULL 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 NULL 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 DN1 column. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the distinct value to the total number of rows. For example, if stadndistinct is -0.5, the actual distinct value is the total number of rows multiplied by 0.5. • The value 0 indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics. This is reserved.

NOTICE

PG_STATISTIC stores sensitive information about statistical objects, such as MCVs. Users with the SYSADMIN permission and authorized users can access the PG_STATISTIC system catalog to query the sensitive information about the statistical objects.

12.2.70 PG_STATISTIC_EXT

PG_STATISTIC_EXT records extended statistics about tables in a database. You can specify the extended statistics to be collected. This system catalog is accessible only to users with the SYSADMIN permission.

Table 12-68 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. • t (true): yes • f (false): no
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. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the distinct value to the total number of rows. For example, if stadistinct is -0.5, the actual distinct value is the total number of rows multiplied by 0.5. • The value 0 indicates that the number of distinct values is unknown.
stadndistinct	real	Number of unique non-null data values in the DN1 column. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the distinct value to the total number of rows. For example, if stadndistinct is -0.5, the actual distinct value is the total number of rows multiplied by 0.5. • The value 0 indicates that the number of distinct values is unknown.
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 NULL 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 NULL 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.

NOTICE

PG_STATISTIC_EXT stores sensitive information about statistical objects, such as MCVs. Users with the SYSADMIN permission and authorized users can access the PG_STATISTIC_EXT system catalog to query the sensitive information about the statistical objects.

12.2.71 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 12-69 PG_SYNONYM columns

Name	Type	Description
oid	oid	Database object ID
synname	name	Synonym name.
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.

12.2.72 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 12-70 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 GRANT and REVOKE .
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 • t (true): yes. • f (false): no.

12.2.73 PG_TRIGGER

PG_TRIGGER records trigger information.

Table 12-71 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 called by the trigger.
tgtype	smallint	Trigger type.
tgenabled	"char"	O : The trigger is triggered in origin or local mode. D : The trigger is disabled. R : The trigger is triggered in replica mode. A : The trigger is always triggered.
tgisinternal	Boolean	Internal trigger ID. If the value is true , it indicates an internal trigger.

Name	Type	Description
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 • t (true): yes. • f (false): no.
tginitdeferred	Boolean	Specifies whether the trigger is of the INITIALLY DEFERRED type. • t (true): yes • f (false): no
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 specified in the trigger.
tgqual	pg_node_tree	WHEN condition of the trigger (NULL if the WHEN condition does not exist).
tgowner	oid	Trigger owner.

12.2.74 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 12-72 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	oid in PG_NAMESPACE	OID of the namespace that contains the configuration

Name	Type	Reference	Description
cfgowner	oid	oid in PG_AUTHID	Owner of the configuration
cfgparser	oid	oid in PG_NAMESPACE	OID of the text search parser for this configuration
cfgoptions	text[]	-	Configuration options

12.2.75 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 12-73 PG_TS_CONFIG_MAP columns

Name	Type	Reference	Description
mapcfg	oid	PG_TS_CONFIG .oid	OID of the PG_TS_CONFIG 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 mapcfg or maptokentype are the same.
mapdict	oid	PG_TS_DICT .oid	OID of the text search dictionary to consult

12.2.76 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 12-74 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	oid in PG_NAMESPACE	OID of the namespace that contains the dictionary
dictowner	oid	oid in PG_AUTHID	Owner of the dictionary
dicttemplate	oid	PG_TS_TEMPLATE .oid	OID of the text search template for the dictionary
dictinitoption	text	-	Initialization option string for the template

12.2.77 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 12-75 PG_TS_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
prsname	name	-	Text search parser name
pramespace	oid	oid in PG_NAMESPACE	OID of the namespace that contains the parser
prsstart	regproc	prname in PG_PROC	Name of the parser's startup function
prstoken	regproc	prname in PG_PROC	Name of the parser's next-token function
prsend	regproc	prname in PG_PROC	Name of the parser's shutdown function
prsheadline	regproc	prname in PG_PROC	Name of the parser's headline function

Name	Type	Reference	Description
prslextype	regproc	prname in PG_PROC	Name of the parser's lextype function

12.2.78 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 12-76 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	oid in PG_NAMESPACE	OID of the namespace that contains the template
tmplinit	regproc	prname in PG_PROC	Name of the template's initialization function
tmpllexize	regproc	prname in PG_PROC	Name of the template's lexize function

12.2.79 PG_TYPE

PG_TYPE stores information about data types.

Table 12-77 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.

Name	Type	Description
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. • -1: a "varlena" type (one that has a length word). • -2: a null-terminated C string.
typbyval	Boolean	• true: A value of this type is passed by value internally. • false: A value of this type is passed by reference internally. If typplen of this type is not 1, 2, 4, or 8, you are advised to transfer a reference value for typbyval. You can also transfer a value for typbyval. Variable-length types are usually passed by reference or by value.
typtype	"char"	• b: basic type. • c: composite type (for example, the row type of a table). • d: domain type. • p: pseudo type. • r: range type. • e: enumeration type. • o: set type. For details, see typrelid and typbasetype.
typcategory	"char"	Fuzzy classification of data types, which can be used by parsers as the basis for data conversion.
typispreferred	Boolean	• true: Data is converted when it complies with the conversion rule specified by typcategory. • false: Data is not converted.
typisdefined	Boolean	• true: The type has been defined. • false: 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.
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.

Name	Type	Description
typrelid	oid	If this is a composite type (see typtype), then this column points to the PG_CLASS entry that defines the corresponding table. For a free-standing composite type, the PG_CLASS entry does not represent a table, but it is required for the type's PG_ATTRIBUTE entries to link to. It is 0 for non-composite type.
typelem	oid	If typelem is not 0 , it identifies another row in PG_TYPE. The current type can be described as an array yielding values of type typelem . A "true" array type has a variable length (typlen = -1), but some fixed-length types (typlen > 0) also have non-zero typelem , for example name and point . If a fixed-length type has a typelem , its internal representation must be a number of values of the typelem 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 0 , 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); 0 for non-input conversion function.
typsend	regproc	Output conversion function (binary format); 0 for non-output conversion function.
typmodin	regproc	Input type modifier function; 0 if none.
typmodout	regproc	Output type modifier function; 0 if none.
typanalyze	regproc	Custom ANALYZE function; 0 if the standard function is used.

Name	Type	Description
typalign	"char"	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: • c: char alignment, that is, no alignment required • s: short alignment (2 bytes on most machines) • i: integer alignment (4 bytes on most machines) • d: double alignment (8 bytes on many machines, but by no means all) NOTICE 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.
typstorage	"char"	typstorage tells for varlena types (those with typlen = -1) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are: • p: Values are always stored plain. • e: Values can be stored in a secondary relationship (if the relation has one, see PG_CLASS.reltoastrelid). • m: Values can be stored compressed inline. • x: Values can be stored compressed inline or stored in secondary storage. NOTICE m domains can also be moved out to secondary storage, but only as a last resort (e and x domains are moved first).
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 typtype), then typbasetype identifies the type that this one is based on. The value is 0 if this type is not a derived type.
typtypmod	integer	Records the typtypmod to be applied to domains' base types by domains (the value is -1 if the base type does not use typmod). This is -1 if this type is not a domain.
typndims	integer	If a field is an array, typndims is the number of array dimensions. This is 0 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 PG_COLLATION 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 NULL if a type has no associated default value. If typdefaultbin is not set to NULL , typdefault must contain a default expression represented by typdefaultbin . If typdefaultbin is NULL and typdefault is not, then typdefault 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.

12.2.80 PG_USER_MAPPING

PG_USER_MAPPING records mappings from local users to remote.

This system catalog is accessible only to users with the SYSADMIN permission. Common users can query the [PG_USER_MAPPINGS](#) view.

Table 12-78 PG_USER_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
umuser	oid	oid in PG_AUTHID	OID of the local role being mapped (0 if the user mapping is public)
umserver	oid	oid in PG_FOREIGN_SERVER	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

12.2.81 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 SYSADMIN permission.

Table 12-79 PG_USER_STATUS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roloid	oid	ID of a role.
failcount	integer	Number of failed attempts.
locktime	timestamp with time zone	Time at which the role is locked.
rolstatus	smallint	Role state. • 0: normal • 1: The role is locked for a specific period of time because the failed login attempts exceed the threshold. • 2: The role is locked by the administrator.
permspac e	bigint	Size of the permanent table storage space used by a role.
tempspac e	bigint	Size of the temporary table storage space used by a role.
password expired	smallint	Specifies whether a password is valid. • 0: The password is valid. • 1: The password is invalid.

12.2.82 PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP provides workload group information in the database.

Table 12-80 PG_WORKLOAD_GROUP columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
workload_gpname	name	Workload group name
respool_oid	oid	ID bound to the resource pool
act_statements	integer	Maximum number of active statements in the workload group

12.2.83 PGXC_CLASS

PGXC_CLASS records replicated or distributed information for each table.

Table 12-81 PGXC_CLASS columns

Name	Type	Description
pcrelid	oid	OID of the table.
pclocatortype	"char"	Locator type. • H: Hash • G: Range • L: List • M: Modulo • N: Round Robin • R: Replication
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 0 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.

12.2.84 PGXC_GROUP

PGXC_GROUP records information about storage node groups.

Table 12-82 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_redistribution	"char"	Specifies whether redistribution is required. Valid value: • n: The node group is not redistributed. • y: The source node group is in redistribution. • t: The destination node group is in redistribution.
group_members	oidvector_ext	Node OID list of the node group.
group_buckets	text	Distributed data bucket group.
is_installation	Boolean	Specifies whether to install a sub-cluster. • t (true): yes • f (false): no
group_acl	aclitem[]	Access permission.
group_kind	"char"	Node group type. The options are as follows: • i: installation node group • n: node group in a common cluster. • e: elastic cluster
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.

12.2.85 PGXC_NODE

PGXC_NODE records information about cluster nodes.

Table 12-83 PGXC_NODE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
node_name	name	Node name.
node_type	"char"	Node type. • C: CN. • D: DN. • S: standby DN.
node_port	integer	Port ID of the node.

Name	Type	Description
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. • t (true): yes • f (false): no
nodeis_primary	Boolean	Specifies whether the current node is preferred to execute non-query operations in the replication table. • t (true): yes • f (false): no
nodeis_preferred	Boolean	Specifies whether the current node is preferred to execute queries in the replication table. • t (true): yes • f (false): no
node_id	integer	Node ID. The value is obtain by calculating the value of node_name 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.)
control_port1	integer	Port used by the TCP proxy communications library of the standby node to listen to the control channel.

Name	Type	Description
nodeis_central	Boolean	Specifies whether the current node is a central control node. It is used only for CNs and is invalid for DNs. • t (true): yes • f (false): no
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. • t (true): yes • f (false): no

12.2.86 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 12-84

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.
pclocatortype	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. i : Table is being redistributed. y : Table redistribution is complete. n : Table has not been redistributed. d : The redistribution is complete, but the temporary table has not been deleted.
redis_orde r	integer	Table redistribution sequence. The default value is 1024 . The value 0 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: 0X0 : The internal mask is disabled. 0X8000 : The internal mask is enabled. 0X01 : The insert operation is not allowed in this table. 0X02 : The delete operation is not allowed in this table. 0X04 : The alter operation is not allowed in this table. 0X08 : The select operation is not allowed in this table. 0X0100 : The update operation is not allowed in this table.
table_size	bigint	Size of a table, in bytes.

12.2.87 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 12-85 PGXC_SLICE columns

Name	Type	Description
relname	name	Table name or shard name, which is distinguished by type .

Name	Type	Description
type	"char"	• 't': relname is the table name. • 's': relname is the shard name.
strategy	"char"	• 'r': range distributed table. • 'l': list distributed table. 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 relname is set to a shard name, nodeoid indicates the OID of the DN where the shard data is stored.
boundar ies	text[]	When relname 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.

12.2.88 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 12-86 PLAN_TABLE_DATA columns

Name	Type	Description
session_id	text	Session that inserts the data. Its value consists of a worker thread start timestamp and a worker thread ID. Values are constrained by NOT NULL .

Name	Type	Description
user_id	oid	User who inserts the data. Values are constrained by NOT NULL .
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 run **gs_clean -C** to manually delete data of inactive sessions.
- 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. To delete data from the table, you are advised to 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.

12.2.89 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.
- 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.

Table 12-87 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
debug_query_id	bigint	ID of the unique SQL statement.	L0
query	text	Normalized SQL (available only on CNs).	L0
start_time	timestamp with time zone	Time when a statement starts.	L0

Name	Type	Description	Record Level
finish_time	timestamp with time zone	Time when a statement ends.	L0
slow_sql_thres hold	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 can be n_soft_parse + n_hard_parse and can be greater than n_calls because the subquery is not counted in n_calls.	L0
n_hard_parse	bigint	Number of hard parsing times. The value can be n_soft_parse + n_hard_parse and can be greater than n_calls because the subquery is not counted in n_calls.	L0
query_plan	text	Statement execution plan.	L1
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement.	L0
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: μ s).	L0

Name	Type	Description	Record Level
cpu_time	bigint	CPU time (unit: μ s).	L0
execution_time	bigint	Execution time in the executor (unit: μ s).	L0
parse_time	bigint	SQL parsing time (unit: μ s).	L0
plan_time	bigint	SQL plan generation time (unit: μ s).	L0
rewrite_time	bigint	SQL rewriting time (unit: μ s).	L0
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μ s).	L0
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μ s).	L0
data_io_time	bigint	I/O time (unit: μ s).	L0
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μ 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: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code> .	L0
net_rcv_info	text	Network status of messages received through a physical connection, including the time (unit: μ 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: <code>{"time":xxx, "n_calls":xxx, "size":xxx}</code> .	L0

Name	Type	Description	Record Level
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μ 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}.	L0
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μ 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}.	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	List of statement lock events, which are recorded in chronological order. The number of records is affected by the track_stmt_details_size parameter. This column is binary and needs to be read using the parsing function pg_catalog.statement_detail_decode . For details, see Table 7-60 . Events include: • Start locking. • Complete locking. • Start lock waiting. • Complete lock waiting. • Start unlocking. • Complete unlocking. • Start lightweight lock waiting. • Complete lightweight lock waiting.	L2
is_slow_sql	Boolean	Specifies whether the SQL statement is a slow SQL statement. • t (true): yes. • f (false): no	L0
trace_id	text	Driver-specific trace ID, which is associated with an application request.	L0
advise	text	Risks which may cause slow SQL statements. (Multiple risks may exist at the same time.) • Cast Function Cause Index Miss.: Index matching may fail due to implicit conversion. • Limit too much rows.: The SQL statement execution may slow down due to a large limit value. • Proleakproof of function is false.: The proleakproof of the function is set to false. In this case, the function does not use statistics when generating a plan due to data leakage risks. As a result, the accuracy of the generated plan is affected and the SQL statement execution may slow down.	L0

12.2.90 STREAMING_STREAM

STREAMING_STREAM records the metadata of all STREAM objects.

Table 12-88 STREAMING_STREAM column

Name	Type	Description
relid	oid	STREAM object ID.
queries	bytea	Bitmap mapping of the CONTVIEW corresponding to the STREAM.

12.2.91 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY records the metadata of all CONTVIEW objects.

Table 12-89 STREAMING_CONT_QUERY columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.
type	"char"	CONTVIEW type. • 'r': CONTVIEW is based on the row-store model.
relid	oid	CONTVIEW object ID.
defrelid	oid	ID of the continuous computing rule view corresponding to CONTVIEW.
active	Boolean	Specifies whether the CONTVIEW is in the continuous computing state. • t (true): yes • f (false): no
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 0 (no overlapping window) and 1 (sliding window, with one step).

Name	Type	Description
ttn	integer	Value of ttn_interval 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.

12.2.92 STREAMING_REAPER_STATUS

STREAMING_REAPER_STATUS records the status information about the reaper thread of the streaming engine. Due to specification changes, the current version no longer supports this feature. Do not use it.

Table 12-90 STREAMING_REAPER_STATUS columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.
contquery_name	name	Name of the CONTVIEW object.
gather_interval	text	Value of gather_interval (time parameter for automatically aggregating historical data before a specific time) set by the CONTVIEW object.
gather_completion_time	text	Time when the latest GATHER (historical data aggregation) of the CONTVIEW object is completed.

12.3 System Views

12.3.1 ADM_COL_COMMENTS

ADM_COL_COMMENTS displays information about table column comments in the database. By default, only users with the SYSADMIN permission 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 12-91 ADM_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.

12.3.2 ADM_CONS_COLUMNS

ADM_CONS_COLUMNS displays information about constraint columns in database tables. By default, only users with the SYSADMIN permission 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 12-92 ADM_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
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.
table_name	character varying(64)	Name of a constraint-related table.

12.3.3 ADM_CONSTRAINTS

ADM_CONSTRAINTS displays information about table constraints in database. By default, only users with the SYSADMIN permission 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 12-93 ADM_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.

Name	Type	Description
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. • c: check constraint. • f: foreign key constraint. • p: primary key constraint. • u: unique constraint.
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).

12.3.4 ADM_DATA_FILES

ADM_DATA_FILES displays the description of database files. By default, only the SYSADMIN 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 12-94 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.

12.3.5 ADM_HIST_SNAPSHOT

ADM_HIST_SNAPSHOT records the index information and start time of WDR snapshots stored in the current system. By default, only the SYSADMIN can access this view. Common users can access the view only after being authorized.

Table 12-95 ADM_HIST_SNAPSHOT columns

Name	Type	Description
snap_id	bigint	WDR snapshot ID.

Name	Type	Description
begin_interval_time	timestamp	Start time of a WDR snapshot.

12.3.6 ADM_HIST_SQL_PLAN

ADM_HIST_SQL_PLAN displays plan information collected by the current user by running the EXPLAIN PLAN statement.

Table 12-96 ADM_HIST_SQL_PLAN columns

Name	Type	Description
sql_id	character varying(30)	Session that inserts the data. Its value consists of a worker thread start timestamp and a worker thread ID. Values are constrained by NOT NULL .
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.

12.3.7 ADM_HIST_SQLSTAT

ADM_HIST_SQLSTAT displays information about statements executed on the current node.

Table 12-97 ADM_HIST_SQLSTAT columns

Name	Type	Description
instance_number	integer	Instance ID of a snapshot.
sql_id	bigint	Query ID.
plan_hash_value	bigint	ID of the normalized SQL statement.
module	text	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: μ s).
cpu_time_delta	bigint	CPU time (unit: μ s).
executions_delta	integer	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: μ 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	integer	Unique snapshot ID.

12.3.8 ADM_IND_COLUMNS

ADM_IND_COLUMNS displays column information about all indexes in the database. By default, only users with the SYSADMIN permission 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 12-98 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.

12.3.9 ADM_IND_EXPRESSIONS

ADM_IND_EXPRESSIONS displays information about expression indexes in the database. By default, only users with the SYSADMIN permission 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 12-99 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.

12.3.10 ADM_IND_PARTITIONS

ADM_IND_PARTITIONS displays information about all index partitions in the database. Each index partition of a partitioned table in the database, if present, has a row of records in ADM_IND_PARTITIONS. By default, only the SYSADMIN 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 12-100 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.
high_value	text	Upper limit of the partition corresponding to the index partition.
index_partition_usable	Boolean	Specifies whether the index partition is available. • t (true): yes. • f (false): no.

Name	Type	Description
schema	character varying(64)	Schema of the partitioned table index to which the index partition belongs.

12.3.11 ADM_INDEXES

ADM_INDEXES displays all indexes in the database. By default, only users with the SYSADMIN permission 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 12-101 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.
partitioned	character(3)	Specifies whether the index has the property of partitioned tables.
generated	character varying(1)	Specifies whether the index name is generated by the system.

12.3.12 ADM_OBJECTS

ADM_OBJECTS displays all database objects in the database. Only the SYSADMIN 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 12-102 ADM_OBJECTS columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.

Name	Type	Description
object_id	oid	Object OID.
object_type	name	Object type. For example, table, schema, and index.
namespace	oid	Namespace containing the object.
created	timestamp with time zone	Creation time of the object.
last_ddl_time	timestamp with time zone	Last modification time of the object.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see [PG_OBJECT](#).

12.3.13 ADM_PART_INDEXES

ADM_PART_INDEXES displays information about all partitioned table indexes in the database. By default, only users with the SYSADMIN permission 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 12-103 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 NOTE Only range partitioning is supported.

Name	Type	Description
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

12.3.14 ADM_PART_TABLES

ADM_PART_TABLES displays information about all partitioned tables in the database. By default, only users with the SYSADMIN permission 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 12-104 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.
partitioning_type	text	Partitioning policy of the partitioned table. NOTE Only range partitioning is supported.
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.

12.3.15 ADM_PROCEDURES

ADM_PROCEDURES displays information about all stored procedures or functions in the database. By default, only users with the SYSADMIN permission 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 12-105 ADM_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function.
object_name	character varying(64)	Name of a stored procedure or function.
argument_number	smallint	Number of input parameters in a stored procedure.

12.3.16 ADM_SCHEDULER_JOBS

ADM_SCHEDULER_JOBS displays information about all DBE_SCHEDULER scheduled jobs in the database.

Table 12-106 ADM_SCHEDULER_JOBS columns

Name	Type	Description
owner	name	Owner of a scheduled job.
job_name	text	Name of a scheduled job.
job_style	text	Action mode of a scheduled job.
job_creator	name	Creator of a scheduled job.
program_name	text	Name of the program referenced by a scheduled job.
job_action	text	Program content of a scheduled job.
number_of_arguments	text	Number of parameters of a scheduled job.
schedule_name	text	Name of the schedule referenced by a scheduled job.
start_date	timestamp without time zone	Start time of a scheduled job.
repeat_interval	text	Period of a scheduled job.
end_date	timestamp without time zone	End time of a scheduled job.

Name	Type	Description
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.
state	"char"	Status of a scheduled job.
failure_count	smallint	Number of scheduled job failures.
last_start_date	timestamp without time zone	Last time when a scheduled job was started.
next_run_date	timestamp without time zone	Next execution time of a scheduled job.
destination	text	Target name of a scheduled job.
credential_name	text	Certificate name of a scheduled job.
comments	text	Comments of a scheduled job.

12.3.17 ADM_SEQUENCES

ADM_SEQUENCES displays information about all sequences in the database. By default, only users with the SYSADMIN permission 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 12-107 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.

Name	Type	Description
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 Y or N . • Y: It is a cyclic sequence. • N: It is not a cyclic sequence.

12.3.18 ADM_SOURCE

ADM_SOURCE displays all stored procedures or functions in the database and provides columns defined by the stored procedures or functions. By default, only users with the SYSADMIN permission 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 12-108 ADM_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function.
name	character varying(64)	Name of a stored procedure or function.
text	text	Definition of a stored procedure or function.

12.3.19 ADM_SYNONYMS

ADM_SYNONYMS displays all synonyms in the database. This view is accessible only to users with SYSADMIN permissions. This view exists in both PG_CATALOG and SYS schemas.

Table 12-109 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 table_owner , 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 table_name , 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 table_schema_name , the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

12.3.20 ADM_TAB_COLUMNS

ADM_TAB_COLUMNS displays information about the columns of tables. Each column of each table in the database has a row of data in ADM_TAB_COLUMNS. By default, only the SYSADMIN 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_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.

Name	Type	Description
data_type	character varying(128)	Data type of the 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 NULL for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and 0 for other data types.
nullable	bpchar	Specifies whether a column can be empty (n for a primary key constraint or NOT NULL constraint).
column_id	integer	Sequence number of the column when the table is created.
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.
comments	text	Comments.

12.3.21 ADM_TAB_COMMENTS

ADM_TAB_COMMENTS displays comments about all tables and views in the database. By default, only users with the SYSADMIN permission 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_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.

12.3.22 ADM_TAB_PARTITIONS

ADM_TAB_PARTITIONS displays information about all partitioned tables in the database. By default, only users with the SYSADMIN permission 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 12-112 ADM_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.
high_value	text	Upper boundary of the range partition and interval partition.
tablespace_name	name	Tablespace name of the partitioned table.
schema	character varying(64)	Name of a namespace.

12.3.23 ADM_TABLES

ADM_TABLES displays information about all partitioned tables in the database. By default, only users with the SYSADMIN permission 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_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. • YES: It is deleted. • NO: It is not deleted.
num_rows	numeric	Estimated number of rows in the table.

Name	Type	Description
status	character varying(8)	Specifies whether the current table is valid. • VALID: The current table is valid. • UNUSABLE: The current table is unavailable.
temporary	character(1)	Specifies whether the table is a temporary table. • Y: The table is a temporary table. • N: The table is not a temporary table.

12.3.24 ADM_TABLESPACES

ADM_TABLESPACES displays information about available tablespaces. By default, only users with the SYSADMIN permission 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_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name

12.3.25 ADM_TRIGGERS

ADM_TRIGGERS displays information about triggers in the database. By default, only the SYSADMIN can access the system catalog. Common users can access the system catalog only after being authorized. This view exists in both PG_CATALOG and SYS schemas.

Table 12-115 ADM_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.

Name	Type	Description
status	character varying(64)	• O: The trigger is enabled in origin or local mode. • D: The trigger is disabled. • R: The trigger is enabled in replica mode. • A: The trigger is always enabled.

12.3.26 ADM_TYPE_ATTRS

ADM_TYPE_ATTRS displays the attributes of the current database object type.

Table 12-116 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 varchar 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 -1 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.
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. • The value -1 indicates a "varlena" type (one that has a length word). • The value -2 indicates a NULL-terminated C string.
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 (c or n).

Name	Type	Description
attr_no	smallint	Attribute number
inherited	character(1)	Specifies whether the attribute is inherited from the super type (Y or N).

12.3.27 ADM_USERS

ADM_USERS displays information about all database users. This view is accessible only to users with SYSADMIN permissions. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-117 ADM_USERS columns

Name	Type	Description
username	character varying(64)	Username.

12.3.28 ADM_VIEWS

ADM_VIEWS displays views in the database. By default, only users with the SYSADMIN permission 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 12-118 ADM_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.
view_name	character varying(64)	View name.

12.3.29 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 users with the SYSADMIN permission can access this view.

Table 12-119 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 debug_query_id .
socket	integer	socket fd 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.

12.3.30 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 12-120 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.

12.3.31 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 12-121 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.

12.3.32 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 12-122 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.

12.3.33 DB_CONSTRAINTS

DB_CONSTRAINTS displays information about constraints accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

Table 12-123 DB_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.

Name	Type	Description
constraint_type	text	Constraint type. • c: check constraint. • f: foreign key constraint. • p: primary key constraint. • u: unique constraint.
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).

12.3.34 DB_DEPENDENCIES

DB_DEPENDENCIES displays dependencies between functions and advanced packages accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

NOTICE

In GaussDB, this table is empty without any record due to information constraints.

Table 12-124 DB_DEPENDENCIES columns

Name	Type	Description
owner	character varying(30)	Object owner.
name	character varying(30)	Object name.
type	character varying(17)	Object type.
referenced_owner	character varying(30)	Owner of a referenced object.
referenced_name	character varying(64)	Name of a referenced object.
referenced_type	character varying(17)	Type of a referenced object.
referenced_link_name	character varying(128)	Name of the link to a referenced object.

Name	Type	Description
schemaid	numeric	ID of the current schema.
dependency_type	character varying(4)	Dependency type (REF indicates soft reference and HARD indicates direct description).

12.3.35 DB_IND_COLUMNS

DB_IND_COLUMNS displays all index columns accessible to the current user. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-125 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

12.3.36 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 12-126 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.
column_expression	text	Function-based index expression of a specified column.

Name	Type	Description
column_position	smallint	Position of the column in the index.

12.3.37 DB_INDEXES

DB_INDEXES displays information about indexes accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

Table 12-127 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 unique.
partitioned	character(3)	Specifies whether the index has the property of partitioned tables.
generated	character varying(1)	Specifies whether the name of an index is generated by the system.

12.3.38 DB_OBJECTS

DB_OBJECTS displays all database objects accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

Table 12-128 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
created	timestamp with time zone	Creation time of the object.
last_ddl_time	timestamp with time zone	Last modification time of the object.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see [PG_OBJECT](#).

12.3.39 DB_PROCEEDURES

DB_PROCEEDURES displays information about all stored procedures or functions accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

Table 12-129 DB_PROCEEDURES columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.

12.3.40 DB_SEQUENCES

DB_SEQUENCES displays all sequences accessible to the current user. This view exists in both PG_CATALOG and SYS schemas.

Table 12-130 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.
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.

Name	Type	Description
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 Y or N . • Y: It is a cyclic sequence. • N: It is not a cyclic sequence.

12.3.41 DB_SOURCE

DB_SOURCE displays information about stored procedures or functions accessible to the current user, and provides columns defined by the stored procedures and functions. This view exists in both PG_CATALOG and SYS schema.

Table 12-131 DB_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	name	Object class.
text	text	Object definition.

12.3.42 DB_SYNONYMS

DB_SYNONYMS displays all synonyms accessible to the current user.

Table 12-132 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 table_owner , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

Name	Type	Description
table_name	text	Name of the associated object. Although the column is called table_name , 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 table_schema_name , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

12.3.43 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.

Table 12-133 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_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 NULL for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and 0 for other data types.
nullable	bpchar	Specifies whether a column can be empty (n for a primary key constraint or NOT NULL constraint).

Name	Type	Description
column_id	integer	Column ID generated when an object is created or a column is added.
char_length	numeric	Column length (in characters) which is valid only for varchar, nvarchar2, bpchar, and char types.
avg_col_len	numeric	Average length of a column, in bytes.
comments	text	Comments.

12.3.44 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 12-134 DB_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.

12.3.45 DB_TABLES

DB_TABLES displays all tables accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-135 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. ● VALID: The current table is valid. ● UNUSABLE: The current table is unavailable.
temporary	character(1)	Specifies whether the table is a temporary table. ● Y: The table is a temporary table. ● N: The table is not a temporary table.
dropped	character varying	Specifies whether the current table is deleted. ● YES: It is deleted. ● NO: It is not deleted.

12.3.46 DB_TRIGGERS

DB_TRIGGERS displays information about triggers accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-136 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)	● O: The trigger is enabled in origin or local mode. ● D: The trigger is disabled. ● R: The trigger is enabled in replica mode. ● A: The trigger is always enabled.

12.3.47 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 users with the SYSADMIN permission can access this view. This view exists in both PG_CATALOG and SYS schema.

Table 12-137 DB_USERS columns

Name	Type	Description
user_id	oid	OID of a user.
username	name	Username.

12.3.48 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 12-138 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.

12.3.49 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS displays the progress of ongoing operations. The view can be accessed only after being authorized.

Table 12-139 DV_SESSION_LONGOPS columns

Name	Type	Description
sid	bigint	OID of the running backend process.
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.

12.3.50 DV_SESSIONS

DV_SESSIONS displays information about all active backend threads. By default, only the SYSADMIN can access this view. Common users can access the view only after being authorized.

Table 12-140 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 0 in GaussDB.
user#	oid	OID of the user logged in to the backend thread. The OID is 0 if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend thread. username is null if the backend thread is a global auxiliary thread. application_name 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;

12.3.51 GET_GLOBAL_PREPARED_XACTS

GET_GLOBAL_PREPARED_XACTS records prepared transactions on all nodes globally.

Table 12-141 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.

12.3.52 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 the initial user, users with the SYSADMIN permission, users with the OPRADMIN permission in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted permissions.

Table 12-142 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 damaged page. It is set to -1 for a non-segment-page table and to a value other than 0 for a segment-page table. This column is used to specify whether a table is a segment-page table during repair.
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 damaged page. The logical path instead of the actual file is displayed for a segment-page table.
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.

12.3.53 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 DN is empty. By default, only the initial user, 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 permissions.

Table 12-143 GLOBAL_BAD_BLOCK_INFO columns

Name	Type	Description
node_name	text	Node information corresponding to the current repaired page clearance result.
result	Boolean	Execution result of clearing repaired pages of the current instance

12.3.54 GLOBAL_COMM_CLIENT_INFO

GLOBAL_COMM_CLIENT_INFO queries information about active client connections of global nodes in a cluster. By default, only the SYSADMIN has the permission to access this system view.

Table 12-144 GLOBAL_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	Client that connects to the current DN, such as a 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 debug_query_id .
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.

12.3.55 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 12-145 GLOBAL_STAT_HOTKEYS_INFO columns

Name	Type	Description
database_name	text	Name of the database where the hotspot key is located.

Name	Type	Description
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 a 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 0.
count	numeric	Frequency of accessing the hotspot key.

12.3.56 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 SYSADMIN or MONADMIN permission.

Table 12-146 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.

Name	Type	Description
local_role	text	Type of the primary node. • UNKNOWN_MODE: The status is unknown. • NORMAL_MODE: single-host node. • PRIMARY_MODE: primary node. • STANDBY_MODE: standby node. • CASCADE_STANDBY_MODE: cascaded standby node. • PENDING_MODE: The node is in the arbitration phase. • RECOVERY_MODE: The node is in the restoration phase. • STANDBY_CLUSTER_MODE: standby cluster node. • MAIN_STANDBY_MODE: main standby cluster node. NOTE The expected values of the primary node type are NORMAL_MODE, PRIMARY_MODE, PENDING_MODE, and RECOVERY_MODE. If other node types are displayed, contact Huawei technical support.

Name	Type	Description
peer_role	text	Type of the standby node. • UNKNOWN_MODE: The status is unknown. • NORMAL_MODE: single-host node. • PRIMARY_MODE: primary node. • STANDBY_MODE: standby node. • CASCADE_STANDBY_MODE: cascaded standby node. • PENDING_MODE: The node is in the arbitration phase. • RECOVERY_MODE: The node is in the restoration phase. • STANDBY_CLUSTER_MODE: standby cluster node. • MAIN_STANDBY_MODE: main standby cluster node. NOTE The expected values of the standby node type are NORMAL_MODE, STANDBY_MODE, CASCADE_STANDBY_MODE, PENDING_MODE, RECOVERY_MODE, STANDBY_CLUSTER_MODE, and MAIN_STANDBY_MODE. If other node types are displayed, contact Huawei technical support.

Name	Type	Description
peer_state	text	Status of the standby node. • UNKNOWN_STATE: The node status is unknown. • NORMAL_STATE: The node is started properly. • NEEDREPAIR_STATE: The current node needs to be repaired. • STARTING_STATE: The node is being started. • WAITING_STATE: The node is waiting to be upgraded. • DEMOTING_STATE: The node is being downgraded. • PROMOTING_STATE: The standby node is being upgraded to the primary node. • BUILDING_STATE: The standby node fails to be started and needs to be rebuilt. • CATCHUP_STATE: The standby node is catching up with the primary node. • COREDUMP_STATE: The node breaks down.
state	text	WAL sender status. • WALSNDSTATE_STARTUP: startup state. • WALSNDSTATE_BACKUP: backup state. • WALSNDSTATE_CATCHUP: catch-up state. • WALSNDSTATE_STREAMING: streaming replication state.
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.

12.3.57 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO displays all Cgroup information in a database.

Table 12-147 GS_ALL_CONTROL_GROUP_INFO columns

Name	Type	Description
name	text	Name of a Cgroup.
type	text	Type of the Cgroup. • GROUP_NONE: no group. • GROUP_TOP: top group. • GROUP_CLASS: class group of the resource, which does not control any thread. • GROUP_BAKWD: backend thread control group. • GROUP_DEFWD: default control group, which controls only the query threads at this level. • GROUP_TSWD: time-sharing control group of each user, which controls the query thread at the bottom layer.
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	Workload Cgroup level.
cpucores	text	Information about the CPU cores used by a Cgroup.

12.3.58 GS_AUDITING

GS_AUDITING displays all audit information about database-related operations. Only the users with SYSADMIN or POLADMIN permission can access this view.

Table 12-148 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 '. • access: DML operations are audited. • privilege: DDL operations are audited.
polenabled	Boolean	Specifies whether to enable the policy. • t (true): enabled. • f (false): disabled.
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the polname 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.

12.3.59 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS displays all audit information about database DML-related operations. Only the users with SYSADMIN or POLADMIN permission can access this view.

Table 12-149 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. • t (true): enabled. • f (false): disabled.

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 polname 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.

12.3.60 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE displays all audit information about database DDL-related operations. Only the users with SYSADMIN or POLADMIN permission can access this view.

Table 12-150 GS_AUDITING_PRIVILEGE columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. The value ' privilege ' indicates that DDL operations are audited.
polenabled	Boolean	Specifies whether to enable the policy. • t (true): enabled. • f (false): disabled.
access_type	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
label_name	name	Resource label name. This column corresponds to the polname 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.

12.3.61 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO displays all DN's resource summaries. This view can be queried only when **enable_dynamic_workload** is set to **on** and the view cannot be executed on DN's. Only the user with SYSADMIN permission can query this view.

Table 12-151 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

12.3.62 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES displays the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 12-152 GS_DB_PRIVILEGES columns

Name	Type	Description
rolename	name	Username
privilege_type	text	ANY permission of a user. For details about the value, see Table 7-114 .
admin_option	text	Whether the ANY permission recorded in the privilege_type column can be re-granted • yes • no

12.3.63 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 12-153 GS_GET_CONTROL_GROUP_INFO columns

Name	Type	Description
group_name	text	Name of a Cgroup.

Name	Type	Description
group_type	text	Type of the Cgroup. • GROUP_NONE: no group. • GROUP_TOP: top group. • GROUP_CLASS: class group of the resource, which does not control any thread. • GROUP_BAKWD: backend thread control group. • GROUP_DEFWD: default control group, which controls only the query threads at this level. • GROUP_TSWD: time-sharing control group of each user, which controls the query thread at the bottom layer.
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.
limits	bigint	Limit of CPUs allocated to the Cgroup.
wdlevel	bigint	Workload Cgroup level.
cpucore	text	Information about the CPU cores used by a Cgroup.
nodegroup	text	Node group name.
group_kind	text	Node group type. The value can be i , n , v , or e . • i: installation node group. • n: node group in a common cluster. • e: elastic cluster.

12.3.64 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 12-154 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.

12.3.65 GS_LABELS

GS_LABELS displays all configured resource labels. Only the users with SYSADMIN or POLADMIN permission can access this view.

Table 12-155 GS_LABELS columns

Name	Type	Description
labelname	name	Resource label name.
labeltype	name	Resource label type This parameter corresponds to the labeltype column in the GS_POLICY_LABEL system catalog.
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.

12.3.66 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 12-156 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.

12.3.67 GS_MASKING

GS_MASKING displays all configured dynamic masking policies. Only the users with SYSADMIN or POLADMIN permission can access this view.

Table 12-157 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

12.3.68 GS_MATVIEWS

GS_MATVIEWS displays information about each materialized view in the database.

Table 12-158 GS_MATVIEWS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE .nspname	Name of the schema of a materialized view.
matviewname	name	PG_CLASS .relname	Name of a materialized view.
matviewowner	name	PG_AUTHID .rolname	Owner of a materialized view.
tablespace	name	PG_TABLESPACE .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).

12.3.69 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 DN, the queries are counted accordingly on the DNs.

Table 12-159 GS_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name.

Name	Type	Description
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: μ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s).

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ s).

12.3.70 GS_STAT_DB_CU

GS_STAT_DB_CU displays CU hits on each node in each database in the cluster. You can clear it using `gs_stat_reset()`. This view can be viewed only by the users with SYSADMIN or MONADMIN permission.

Table 12-160 GS_STAT_DB_CU columns

Name	Type	Description
node_name1	text	Node name
db_name	text	Database name
mem_hit	bigint	Number of memory hits
hdd_sync_read	bigint	Number of synchronous hard disk reads
hdd_asyn_read	bigint	Number of asynchronous hard disk reads

12.3.71 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU queries the CU hit rate of running sessions on each node in the entire cluster. This data about a session is cleared when you exit this session. After the cluster is restarted, the statistics are also cleared. This view can be viewed only by the users with SYSADMIN or MONADMIN permission.

Table 12-161 GS_STAT_SESSION_CU columns

Name	Type	Description
node_name1	text	Node name

Name	Type	Description
mem_hit	integer	Number of memory hits
hdd_sync_read	integer	Number of synchronous hard disk reads
hdd_asyn_read	integer	Number of asynchronous hard disk reads

12.3.72 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL returns the memory usage (in MB) of the current node group of the database. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 12-162 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Node group name.
memorytype	text	Memory type. The value must be one of the following: • ng_total_memory: total memory size of the node group. • ng_used_memory: memory used by the node group. • ng_estimate_memory: estimated memory used by the node group. • ng_foreignrp_memsize: total memory of the external resource pool of the node group. • ng_foreignrp_usesize: memory used by the external resource pool of the node group. • ng_foreignrp_peaksize: peak memory used by the external resource pool of the node group. • ng_foreignrp_mempct: percentage of the external resource pool of the node group to the total memory of the node group. • ng_foreignrp_estmsize: estimated memory used by the external resource pool of the node group.
memorybytes	integer	Size of allocated memory-typed memory.

12.3.73 GV_SESSION

GV_SESSION view describes the information related to the current user's queries. The columns save information about the last query.

Table 12-163 GV_SESSION columns

Name	Type	Description
sid	bigint	Session ID.
serial#	integer	Sequence number of the active backend thread, which is 0 in GaussDB.
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 0 if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend thread. username is null if the backend thread is a global auxiliary thread.
machine	text	Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will be non-null only for IP connections and only when log_hostname is enabled.
sql_id	bigint	ID of a query.
client_info	text	Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will be non-null only for IP connections and only when log_hostname is enabled.
event	text	Queuing status of a statement. Its value can be: • waiting in queue: The statement is in the queue. • Empty: The statement is running.
sql_exec_start	timestamp with time zone	Time when the currently active query was started, or if state is not active , when the last query was started.
program	text	Name of the application connected to the backend.

Name	Type	Description
status	text	Overall status of this backend. Its value can be: • active: The backend is executing a query. • idle: The backend is waiting for a new client command. • idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction. • idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction. • fastpath function call: The backend is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in this backend.

12.3.74 MPP_TABLES

MPP_TABLES displays information about tables in PGXC_CLASS.

Table 12-164 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.

12.3.75 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 12-165 MY_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.

12.3.76 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 12-166 MY_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
table_name	character varying(64)	Name of a constraint-related table.
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.

12.3.77 MY_CONSTRAINTS

MY_CONSTRAINTS displays table constraint information accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-167 MY_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	vcharacter varying(64)	Constraint name.

Name	Type	Description
constraint_type	text	Constraint type. • c: check constraint. • f: foreign key constraint. • p: primary key constraint. • u: unique constraint.
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).

12.3.78 MY_IND_COLUMNS

MY_IND_COLUMNS displays column information about all indexes accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-168 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.

12.3.79 MY_IND_EXPRESSIONS

MY_IND_EXPRESSIONS displays information about function-based expression indexes accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-169 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.

12.3.80 MY_IND_PARTITIONS

MY_IND_PARTITIONS displays information about index partitions accessible to the current user. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-170 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.
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.
index_partition_usable	Boolean	Specifies whether the index partition is available. • t (true): yes. • f (false): no.
schema	character varying(64)	Schema of the partitioned table index to which the index partition belongs.

12.3.81 MY_INDEXES

MY_INDEXES displays index information about the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-171 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 unique.
partitioned	character(3)	Specifies whether the index has the property of partitioned tables.
generated	character varying(1)	Specifies whether the index name is generated by the system.

12.3.82 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 12-172 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.
start_date	timestamp without time zone	Job start time.
start_suc	text	Start time of the 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
this_date	timestamp without time zone	Start time of an ongoing job execution.
this_suc	text	Start time of the ongoing successful job execution.
next_date	timestamp without time zone	Schedule time of the next job execution.
next_suc	text	Schedule time of the next successful job execution.
broken	text	If the value of the status 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'. • r: running • s: successful execution • f: failed execution • d: canceling an execution
interval	text	Time expression used to calculate the next time the job will be executed. If this parameter is set to null , 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.

12.3.83 MY_OBJECTS

MY_OBJECTS displays database objects accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-173 MY_OBJECTS columns

Name	Type	Description
object_name	name	Object name.
object_id	oid	Object OID.

Name	Type	Description
object_type	name	Object type, including TABLE, INDEX, SEQUENCE, and VIEW.
namespace	oid	Namespace that the object belongs to.
created	timestamp with time zone	Creation time of the object.
last_ddl_time	timestamp with time zone	Last modification time of the object.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see [PG_OBJECT](#).

12.3.84 MY_PART_INDEXES

MY_PART_INDEXES displays information about partitioned table indexes accessible to the current user. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-174 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 NOTE Only range partitioning is supported.
schema	character varying(64)	Schema of the partitioned table index

Name	Type	Description
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs

12.3.85 MY_PART_TABLES

MY_PART_TABLES displays information about partitioned tables accessible to the current user. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-175 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. NOTE Only range partitioning is supported.
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.

12.3.86 MY_PROCEDESURES

MY_PROCEDESURES displays information about stored procedures or functions owned by the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-176 MY_PROCEDESURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function.

Name	Type	Description
object_name	character varying(64)	Name of a stored procedure or function.
argument_number	smallint	Number of input parameters in a stored procedure.

12.3.87 MY_SEQUENCES

MY_SEQUENCES displays the sequence information about the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-177 MY_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.
increment_by	int16	Value by which the sequence is incremented.
cycle_flag	character(1)	Specifies whether a sequence is a cyclic sequence. The options are as follows: • Y: It is a cyclic sequence. • N: It is not a cyclic sequence.
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.

12.3.88 MY_SOURCE

MY_SOURCE displays information about stored procedures or functions owned by the current user and provides columns defined by the stored procedures or the functions. This view exists in both PG_CATALOG and SYS schema.

Table 12-178 MY_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function.
name	character varying(64)	Name of a stored procedure or function.
text	text	Definition of a stored procedure or function.

12.3.89 MY_SYNONYMS

MY_SYNONYMS displays synonyms accessible to the current user.

Table 12-179 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 table_owner , 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 table_name , 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 table_schema_name , its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

12.3.90 MY_TAB_COLUMNS

MY_TAB_COLUMNS displays information about table columns accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-180 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_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 NULL for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and 0 for other data types.
nullable	bpchar	Specifies whether a column can be empty (n for a primary key constraint or NOT NULL constraint).
column_id	integer	Sequence number of the column when the table is created.
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.
comments	text	Comments.

12.3.91 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 12-181 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.

12.3.92 MY_TAB_PARTITIONS

MY_TAB_PARTITIONS displays all table partitions accessible to the current user. Each table partition of a partitioned table accessible to the current user has one record in USER_TAB_PARTITIONS. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-182 MY_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of a partitioned table.
partition_name	character varying(64)	Partition name.
high_value	text	Upper boundary of the table partition.
tablespace_name	name	Tablespace name of a partition.
schema	character varying(64)	Schema of a partitioned table.

12.3.93 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 12-183 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. • yes: deleted. • no: not deleted.
num_rows	numeric	Estimated number of rows in the table.
status	character varying(8)	Specifies whether the current record is valid. • valid: valid.
temporary	character(1)	Specifies whether the table is a temporary table. • y: The table is a temporary table. • n: The table is not a temporary table.

12.3.94 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 12-184 MY_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name.
table_name	character varying(64)	Relational table name.
table_owner	character varying(64)	Role name.
status	character varying(64)	Trigger status. • O: The trigger is enabled in origin or local mode. • D: The trigger is disabled. • R: The trigger is enabled in replica mode. • A: The trigger is always enabled.

12.3.95 MY_VIEWS

MY_VIEWS displays information about all views of the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-185 MY_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.
view_name	character varying(64)	View name.

12.3.96 PG_COMM_DELAY

PG_COMM_DELAY displays the communications library delay status for a single DN.

Table 12-186 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
min_delay	integer	Minimum delay of the current physical connection within 1 minute, in microsecond NOTE 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

12.3.97 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 12-187 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 • UNKNOWN: The logical connection status is unknown. • READY: The logical connection is ready. • RUN: The logical connection sends packets normally. • HOLD: The logical connection is waiting to send packets. • CLOSED: The logical connection is closed. • TO_CLOSED: The logical connection will be closed.
query_id	bigint	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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

12.3.98 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 12-188 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
state	text	Status of the stream • UNKNOWN: The logical connection status is unknown. • READY: The logical connection is ready. • RUN: The logical connection sends packets normally. • HOLD: The logical connection is waiting to send packets. • CLOSED: The logical connection is closed. • TO_CLOSED: The logical connection will be closed.
query_id	bigint	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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

Name	Type	Description
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

12.3.99 PG_COMM_STATUS

PG_COMM_STATUS displays the communications library status for a single DN.

Table 12-189 PG_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 bigint node, in KB/s
txkbyte_rate	bigint	Sending rate of the communications library on the bigint node, in KB/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the libcomm process, in bytes
memkbyte_libpq	bigint	Communication memory size of the libpq process, in bytes
used_pm	integer	Real-time usage of the postmaster thread
used_sflow	integer	Real-time usage of the gs_sender_flow_controller thread
used_rflow	integer	Real-time usage of the gs_receiver_flow_controller thread
used_rloop	integer	Highest real-time usage among multiple gs_receivers_loop threads
stream	integer	Total number of used logical connections.

12.3.100 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 12-190 PG_CONTROL_GROUP_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the Cgroup

12.3.101 PG_CURSORS

PG_CURSORS displays cursors that are currently available.

Table 12-191 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 TRUE . Otherwise, the value is FALSE .
is_binary	Boolean	Specifies whether the cursor is declared BINARY. If it is, the value is TRUE . Otherwise, the value is FALSE .
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 TRUE . Otherwise, the value is FALSE .
creation_time	timestamp with time zone	Timestamp at which the cursor is declared.

12.3.102 PG_EXT_STATS

PG_EXT_STATS displays extended statistics stored in [PG_STATISTIC_EXT](#).

Table 12-192 PG_EXT_STATS columns

Name	Type	Reference	Description
schemaname	name	nspname in PG_NAMESPACE	Name of the schema that contains a table.
tablename	name	relname in PG_CLASS	Table name.
attname	int2vector	stakey in PG_STATISTIC_EXT	Columns to be combined for collecting statistics
inherited	Boolean	-	Table inheritance is not supported currently. The value of this column is false .
null_frac	real	-	Percentage of column combinations that are null to all records.
avg_width	integer	-	Average width of column combinations, in bytes.
n_distinct	real	-	- Estimated number of distinct values in a column combination if the value is greater than 0. - 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. - The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. - The positive form is used when the column seems to have a fixed number of possible values. - The number of distinct values is unknown if the value is 0.

Name	Type	Reference	Description
n_dndistinct	real	-	Number of not-null distinct values in the dn1 column combination. - Exact number of distinct values if the value is greater than 0. - 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, n_dndistinct equals -0.5. - The number of distinct values is unknown if the value is 0.
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 NULL. None of the most common values in the column is NULL.
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 most_common_vals is NULL, the value of this column is NULL.
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 NULL. At least one of the common values in the column is NULL.

Name	Type	Reference	Description
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 most_common_vals_null is NULL , the value of this column is NULL .
histogram_bounds	anyarray	-	Boundary value list of the histogram.

12.3.103 PG_GET_INVALID_BACKENDS

The PG_GET_INVALID_BACKENDS view displays information about current standby DN backend threads connected to the CN. Only users with the SYSADMIN or MONADMIN permission can access this view.

Table 12-193 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

12.3.104 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 12-194 PG_GET_SENDERS_CATCHUP_TIME columns

Name	Type	Description
pid	bigint	Current sender thread ID.

Name	Type	Description
lwpid	integer	Current sender lwpid.
local_role	text	Local role.
peer_role	text	Peer role.
state	text	Current sender's replication status
type	text	Current sender type
catchup_start	timestamp with time zone	Startup time of a catchup task.
catchup_end	timestamp with time zone	End time of the catchup task.

12.3.105 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between group members.

Table 12-195 PG_GROUP columns

Name	Type	Description
groname	name	Group name
grosysid	oid	Group ID
grolist	oid[]	An array, including all the role IDs in this group.

12.3.106 PG_INDEXES

PG_INDEXES displays information about each index in the database.

Table 12-196 PG_INDEXES columns

Name	Type	Reference	Description
schemaname	name	nspname in PG_NAMESPACE	Name of the schema that contains tables and indexes.
tablename	name	relname in PG_CLASS	Name of the table for which the index serves.

Name	Type	Reference	Description
indexname	name	relname in PG_CLASS	Index name
tablespace	name	nspname in PG_TABLESPACE	Name of the tablespace that contains the index.
indexdef	text	-	Index definition (a reconstructed CREATE INDEX command).

12.3.107 PG_LOCKS

PG_LOCKS displays information about locks held by open transactions.

Table 12-197 PG_LOCKS columns

Name	Type	Reference	Description
locktype	text	-	Type of the locked object. The value can be relation , extend , page , tuple , transactionid , virtualxid , object , userlock , or advisory .
database	oid	oid in PG_DATABASE	OID of the database in which the locked target exists. - The OID is 0 if the target is a shared object. - The OID is NULL if the locked object is a transaction.
relation	oid	oid in PG_CLASS	OID of the relationship targeted by the lock. The value is NULL if the object is not a relationship or part of a relationship.
page	integer	-	Page number targeted by the lock within the relationship (NULL if the object is not a relation page or row page).
tuple	smallint	-	Row number targeted by the lock within the page (NULL if the object is not a row).
bucket	integer	-	Hash bucket ID.
virtualxid	text	-	ID of the virtual transaction The value is NULL if the object is not a virtual transaction.

Name	Type	Reference	Description
transactionid	xid	-	ID of the transaction. The value is NULL if the object is not a transaction.
classid	oid	oid in PG_CLASS	OID of the system catalog that contains the object (NULL if the object is not a general database object).
objid	oid	-	OID of the locked object within its system catalog. The value is NULL if the object is not a general database object.
objsubid	smallint	-	Column number for a column in the table (0 if the target is of other object type and NULL if the target is not a general database object).
virtualtransaction	text	-	ID of the virtual transaction holding or awaiting this lock.
pid	bigint	-	ID of the server logic thread that holds or waits for the lock. The value is NULL 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	-	- The value is TRUE if the lock is a held lock. - The value is FALSE if the lock is an awaited lock.
fastpath	Boolean	-	The value is TRUE if the lock is obtained through fast-path , and is FALSE 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.

12.3.108 PG_NODE_ENV

PG_NODE_ENV displays the environment variable information of the current node. Only the SYSADMIN or MONADMIN can access the system view.

Table 12-198 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.

12.3.109 PG_OS_THREADS

PG_OS_THREADS provides status information about all the threads under the current node.

Table 12-199 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

12.3.110 PG_POOLER_STATUS

PG_POOLER_STATUS queries the cache connection status in the pooler.

Table 12-200 PG_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	Specifies whether the connection is currently used. • t (true): The connection is in use. • f (false): The connection is not in use.
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.

PG_POOLER_STATUS can only perform the query on the CN and display connection cache information about the pooler module.

12.3.111 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays information about all prepared statements that are available in the current session.

Table 12-201 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 regtype . The OID corresponding to an element of this array can be obtained by converting the regtype value to an OID value.
from_sql	Boolean	- Its value is true if the prepared statement was created through the PREPARE statement - Its value is false if the statement was prepared through the frontend/backend protocol

12.3.112 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 12-202 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	rolname in PG_AUTHID	Name of the user who executes the transaction.
database	name	datname in PG_DATABASE	Name of the database that executes the transaction.

12.3.113 PG_REPLICATION_ORIGIN_STATUS

PG_REPLICATION_ORIGIN_STATUS displays the replication status of the replication source.

Table 12-203 PG_REPLICATION_ORIGIN_STATUS columns

Name	Type	Description
local_id	oid	Replication source ID.
external_id	text	Name of the replication source.
remote_lsn	text	Remote LSN of the replication source.
local_lsn	text	Local LSN of the replication source.

12.3.114 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS displays replication slot information.

Table 12-204 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. • physical: physical replication slot. • logical: logical replication slot.
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. • t (true): yes. • f (false): no.
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	Lab features.
confirmed_flush	text	Dedicated logical replication slot. The client confirms the location of the received log.

12.3.115 PG_RLSPOLICIES

PG_RLSPOLICIES displays information about row-level security policies. The initial user and users with the SYSADMIN attribute can view all policy information. Other users can view only the policy information in their own tables.

Table 12-205 PG_RLSPOLICIES columns

Name	Type	Description
schemaname	name	Name of the schema of the table object to which a row-level security policy is applied.
tablename	name	Name of the table object to which the row-level security policy is applied.
polycyname	name	Name of a row-level security policy.
policypermissive	text	Expression concatenation mode of a row-level security policy. Value range: ● PERMISSIVE: permissive policy, which is concatenated using the OR expression. ● RESTRICTIVE: restrictive policy, which is concatenated using the AND expression.
policyroles	name[]	List of users affected by the row-level security policy. If this parameter is not specified, all users are affected.
polycycmd	text	SQL operations affected by the row-level security policy.
policyqual	text	Expression of a row-level security policy.

12.3.116 PG_ROLES

PG_ROLES displays information about database roles. The initial user and users with the SYSADMIN or CREATEROLE attribute can view information about all roles. Other users can view only their own information.

Table 12-206 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. • t (true): yes. • f (false): no.
rolinherit	Boolean	N/A	Specifies whether the role inherits the permissions for this type of roles. • t (true): yes. • f (false): no.
rolcreatorole	Boolean	N/A	Specifies whether the role can create other roles. • t (true): yes. • f (false): no.
rolcreatedb	Boolean	N/A	Specifies whether the role can create databases. • t (true): yes. • f (false): no.
rolcatupdate	Boolean	N/A	Specifies whether the role can update system catalogs directly. Only the initial system administrator whose usesysid is 10 has this permission. This permission is unavailable for other users. • t (true): yes. • f (false): no.
rolcanlogin	Boolean	N/A	Specifies whether the role can log in to the database. • t (true): yes. • f (false): no.

Name	Type	Reference	Description
rolreplication	Boolean	N/A	Specifies whether the role can be replicated. • t (true): yes. • f (false): no.
rolauditadmin	Boolean	N/A	Specifies whether the role is an AUDITADMIN. • t (true): yes. • f (false): no.
rolsystemadmin	Boolean	N/A	Specifies whether the role is a SYSADMIN. • t (true): yes. • f (false): no.
rolconnlimit	integer	N/A	Maximum number of concurrent connections that this role can initiate if this role can log in. The value -1 indicates no limit.
rolpassword	text	N/A	Encrypted user password. The value is displayed as ***** .
rolvalidbegin	timestamp with time zone	N/A	Start time for account validity (NULL if start time is not specified).
rolvaliduntil	timestamp with time zone	N/A	End time for account validity (NULL if end time is not specified).
rolparentid	oid	rolparentid in PG_AUTHID	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[]	setconfig in PG_DB_ROLE_SETTING	Default value of runtime configuration items.
oid	oid	oid in PG_AUTHID	Role ID

Name	Type	Reference	Description
roluseft	Boolean	roluseft in PG_AUTHID	Specifies whether the role can perform operations on foreign tables • t (true): yes. • f (false): no.
rolkind	"char"	N/A	Role type • n: common user, that is, non-permanent user. • p: permanent user.
nodegroup	name	N/A	Name of the node group associated with a role. If no node group is associated, leave it blank.
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.
rolmonitorad min	Boolean	N/A	Specifies whether the role is MONADMIN. • t (true): yes. • f (false): no.
roloperatorad min	Boolean	N/A	Specifies whether the role is an O&M administrator. • t (true): yes. • f (false): no.
rolpolicyadmi n	Boolean	N/A	Specifies whether the role is a POLADMIN. • t (true): yes. • f (false): no.

12.3.117 PG_RULES

PG_RULES is used to query useful information about rewriting rules.

Table 12-207 PG_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in.
tablename	name	Name of the table to which the rule applies.

Name	Type	Description
rulename	name	Name of a rule.
definition	text	Rule definition (reconstructed by the CREATE statement).

12.3.118 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays information about running transactions on the current node.

Table 12-208 PG_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status. • 3: prepared. • 0: starting.
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum. • t (true): yes. • f (false): no.
timeline	bigint	Number of database restarts.
prepare_xid	xid	ID of the transaction in the prepared state; otherwise, the value is 0 .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent from a CN to a DN.

12.3.119 PG_SECLABELS

PG_SECLABELS provides information about security labels.

Table 12-209 PG_SECLABELS columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to.
classoid	oid	oid in PG_CLASS	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 (by referring to objoid and classoid in the table). - For all other object types, this column is 0.
objtype	text	-	Type of the object to which the label belongs, in text format. Example: table: table type. column: column type.
objnamespace	oid	oid in PG_NAMESPACE	OID of the namespace for the object, if applicable; otherwise, NULL .
objname	text	-	Name of the object to which the label belongs, in text format.
provider	text	provider in PG_SECLABEL	Provider of the label.
label	text	label in PG_SECLABEL	Security label name.

12.3.120 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 12-210 PG_SETTINGS columns

Name	Type	Description
name	text	Parameter name.
setting	text	Current parameter value.
unit	text	Unit of the parameter.

Name	Type	Description
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 internal , postmaster , sighup , backend , superuser , and user .
vartype	text	Parameter type, including bool , enum , integer , real , or string .
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 null .
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is null .
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is null .
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 null .
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 null .

12.3.121 PG_SHADOW

PG_SHADOW displays the attributes of all roles marked with rolcanlogin in **PG_AUTHID**. Only users with the SYSADMIN permission 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 12-211 PG_SHADOW columns

Name	Type	Reference	Description
username	name	rolname in PG_AUTHID	Username.
usesysid	oid	oid in PG_AUTHID	ID of this user.
usecreatedb	Boolean	-	Specifies whether a user has the permissions to create databases. • t (true): yes. • f (false): no.
usesuper	Boolean	-	Specifies whether the user is a SYSADMIN. • t (true): yes. • f (false): no.
usecatupd	Boolean	-	Specifies whether the user can update a view. Even the SYSADMIN cannot update the view unless this column is t . • t (true): yes. • f (false): no.
userepl	Boolean	-	Specifies whether the user has the permissions to duplicate data streams. • t (true): yes. • f (false): no.
passwd	text	rolpassword in PG_AUTHID	Password ciphertext. If there is no password, the value is NULL .
valbegin	timestamp with time zone	-	Start time for account validity (NULL if start time is not specified).
valuntil	timestamp with time zone	-	End time for account validity (NULL 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[]	setconfig in PG_DB_ROLE_SETTING	Default value of runtime configuration items.
tempspaceli mit	text	-	Storage space of the temporary table, in KB.
spillspaceli mit	text	-	Operator disk flushing space, in KB.
usemonitor admin	Boolean	-	Specifies whether the user has the MONADMIN permission. • t (true): yes. • f (false): no.
useoperator admin	Boolean	-	Specifies whether the user has OPRADMIN permission. • t (true): yes. • f (false): no.
usepolicyad min	Boolean	-	Specifies whether the user has the POLADMIN permission. • t (true): yes. • f (false): no.

12.3.122 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL displays usage information about all the shared memory contexts.

Table 12-212 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.

12.3.123 PG_STATS

PG_STATS displays single-column statistics stored in the pg_statistic table. The **autovacuum_naptime** parameter specifies the interval for updating statistics recorded in the view.

Table 12-213 PG_STATS columns

Name	Type	Reference	Description
schemaname	name	nspname in PG_NAMESPACE	Name of the schema that the table is in.
tablename	name	relname in PG_CLASS	Table name.
attname	name	attname in PG_ATTRIBUTE	Column name.
inherited	boolean	-	Table inheritance is not supported currently. The value of this column is false .
null_frac	real	-	Percentage of column entries that are null.
avg_width	integer	-	Average width in bytes of column's entries.

Name	Type	Reference	Description
n_distinct	real	-	- Estimated number of distinct values in the column if the value is greater than 0. - 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. - The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. - The positive form is used when the column seems to have a fixed number of possible values. - The number of distinct values is unknown if the value is 0.
n_dndistinct	real	-	Number of not-null distinct values in the dn1 column. - Exact number of distinct values if the value is greater than 0. - Negative of the number of distinct values divided by the number of rows if the value is less than 0. For example, if the value of a column appears twice in average, set n_dndistinct to -0.5. - The number of distinct values is unknown if the value is 0.
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 NULL .

Name	Type	Reference	Description
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. The value is NULL if the value of most_common_vals is NULL .
histogram_bounds	anyarray	-	Frequency histogram consisting of values excluding null values and MCVs. If a value appears in the value of most_common_vals , it does not appear in the histogram. If the column data type does not have the < operator or the list specified by most_common_vals contains all values of the column, the histogram information of the column is NULL .
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 NULL 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.

12.3.124 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY shows information about the current user's queries. The columns save information about the last query.

Table 12-214 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 null , 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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 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 (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 query_start column.

Name	Type	Description
query_start	timestamp with time zone	Time when the current active query was started. If state is not active , when the last query was started. For a stored procedure, function, or package, 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 state 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 true . Otherwise, the value is false .
enqueue	text	Queuing status of a statement. Its value can be: • waiting in queue: The statement is in the queue. • Empty: The statement is running.

Name	Type	Description
state	text	Overall status of this backend. Its value can be: • active: The backend is executing a query. • idle: The backend is waiting for a new client command. • idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction. • idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction. • fastpath function call: The backend is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in this backend. NOTE Common users can view their own session status only. The state information of other accounts is empty. For example, after user judy is connected to the database, the state information of user joe and the initial user omm in <code>pg_stat_activity</code> is empty. <pre>SELECT datname, username, usesysid, state,pid FROM pg_stat_activity; datname username usesysid state pid -----+-----+-----+-----+----- testdb omm 10 idle 139968752121616 testdb omm 10 idle 139968903116560 db_tpcds judy 16398 active 139968391403280 testdb omm 10 idle 139968643069712 testdb omm 10 idle 139968680818448 testdb joe 16390 idle 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 state is active , 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.
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.

12.3.125 PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG displays information about all queries in the node group of the current user.

Table 12-215 PG_STAT_ACTIVITY_NG 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.
global_sessionid	text	Global 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.

Name	Type	Description
client_addr	inet	IP address of the client connected to the backend. If this column is NULL , it indicates that either the client is connected via a UDS 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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 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 (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 query_start column.
query_start	timestamp with time zone	Time when the current active query was started. If state is not active , when the last query was started. For a stored procedure, function, or package, 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 state 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 true . Otherwise, the value is false .
enqueue	text	Queuing status of a statement. The value must be one of the following: • waiting in queue: The statement is in the queue. • Empty: The statement is running.

Name	Type	Description
state	text	Overall status of this backend. The value must be one of the following: • active: The backend is executing a query. • idle: The backend is waiting for a new client command. • idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction. • idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction. • fastpath function call: The backend is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in this backend. NOTE Common users can view their own session status only. The state information of other accounts is empty. For example, after user judy is connected to the database, the state information of user joe and the initial user omm in <code>pg_stat_activity</code> is empty. <pre>SELECT datname, username, usesysid, state,pid FROM pg_stat_activity_ng; 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 state is active , this column shows the ongoing query. In all other states, it shows the last query that was executed.
node_group	text	Node group of the user to which the statement belongs.

12.3.126 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 12-216 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.
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 table rows fetched by simple index scans using the index.

12.3.127 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 accesses to that specific table.

Table 12-217 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.
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.
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.

Name	Type	Description
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 at which the table was updated (by INSERT/UPDATE/DELETE or EXCHANGE/TRUNCATE/DROP PARTITION). This column is not recorded in the system catalog but is recorded on the local CN.

12.3.128 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK shows statistics about Page or CU verification failures after a node is started.

Table 12-218 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: • 0: main data file. • 1: FSM file. • 2: VM file. • 3: BCM file.
error_count	integer	Number of verification failures.

Name	Type	Description
first_time	timestamp with time zone	Time of the first verification failure.
last_time	timestamp with time zone	Time of the latest verification failure.

12.3.129 PG_STAT_BGWRITER

PG_STAT_BGWRITER displays statistics about the activity of the background writer process.

Table 12-219 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.
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.

Name	Type	Description
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.

12.3.130 PG_STAT_DATABASE

PG_STAT_DATABASE displays statistics about each database in the cluster.

Table 12-220 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.
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

Name	Type	Description
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 PG_STAT_DATABASE_CONFLICTS .
temp_files	bigint	Number of temporary files created by queries in this database. Calculates all temporary files, regardless of why the temporary files are created (such as sorting or hashing) or the setting of the log_temp_files parameter.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. Calculates all temporary files, regardless of why the temporary files are created or the setting of the log_temp_files parameter.
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).
stats_reset	timestamp with time zone	Time when the status statistics are reset.

12.3.131 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 12-221 PG_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	ID of a database

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 conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

12.3.132 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 12-222 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.
backend_start	timestamp with time zone	Start time of the program.

Name	Type	Description
state	text	Status of the log synchronization thread: • startup: The thread is being started. • catchup: The thread is establishing a connection between the standby node and the primary node. • streaming: The thread has established a connection between the standby node and the primary node and is replicating data streams. • backup: The thread is sending a backup. • stopping: The thread is being stopped.
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 (0 indicates asynchronization).

Name	Type	Description
sync_state	text	Synchronization state: • async: asynchronous replication. • sync: synchronous replication. • potential: The standby node is asynchronous currently, but if a current synchronization node fails, the standby node becomes synchronous. • quorum: switches between synchronous and asynchronous state to ensure that there are more than a certain number of synchronous standby nodes. Generally, the number of synchronous standby nodes is $(n+1)/2-1$, where n 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 synchronous_standby_names parameter.

12.3.133 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays index status information about all the system catalogs in pg_catalog and information_schema.

Table 12-223 PG_STAT_SYS_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 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 table rows fetched in the original table by a simple index scan that uses the index.

12.3.134 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 12-224 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.

12.3.135 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 12-225 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

12.3.136 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 12-226 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 table rows fetched in the original table by a simple index scan that uses the index.

12.3.137 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays status information about user-defined ordinary tables and TOAST tables in the namespaces.

Table 12-227 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 VACUUM FULL).
last_autovacuum	timestamp with time zone	Last time at which 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 thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting VACUUM FULL).

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_changed	timestamp with time zone	Last modification time of the table data.

12.3.138 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 12-228 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)

12.3.139 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays transaction status information of the system catalog in the namespace.

Table 12-229 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)

12.3.140 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS contains statistics on the execution of each function.

Table 12-230 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

12.3.141 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays transaction status information of the user table in the namespace.

Table 12-231 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)

12.3.142 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 12-232 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.

Name	Type	Description
idx_blks_read	bigint	Number of disk blocks read from this index.
idx_blks_hit	bigint	Number of cache hits in this index.

12.3.143 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES displays the I/O statistics of each sequence in the current database.

Table 12-233 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.

12.3.144 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 12-234 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.

Name	Type	Description
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.

12.3.145 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays I/O status information about all system catalog indexes in the namespaces.

Table 12-235 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.

12.3.146 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays I/O status information about all the sequences in the namespaces.

Table 12-236 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.

12.3.147 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES shows I/O status information about all the system catalogs in the namespaces.

Table 12-237 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.
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.

12.3.148 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays I/O status information about all the user relationship table indexes in the namespaces.

Table 12-238 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.

12.3.149 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES shows I/O status information about all the user relationship table sequences in the namespaces.

Table 12-239 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.
blks_read	bigint	Number of disk blocks read from this sequence.
blks_hit	bigint	Number of cache hits in this sequence.

12.3.150 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays I/O status information about all the user relationship tables in the namespaces.

Table 12-240 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.

12.3.151 PG_TABLES

PG_TABLES is used to query useful information about each table in a database.

Table 12-241 PG_TABLES columns

Name	Type	Reference	Description
schemaname	name	nspname in PG_NAMESPACE	Name of the schema that contains a table.
tablename	name	relname in PG_CLASS	Table name.
tableowner	name	pg_get_userbyid(relowner in PG_CLASS)	Table owner.
tablespace	name	spcname in PG_TABLESPACE	Tablespace that contains the table (default value: NULL).

Name	Type	Reference	Description
hasindexes	Boolean	relhasindex in PG_CLASS	Specifies whether the table has (or recently had) an index. If it does, the value is TRUE . Otherwise, the value is FALSE .
hasrules	Boolean	relhasrules in PG_CLASS	Specifies whether the table has rules. If it does, the value is TRUE . Otherwise, the value is FALSE .
hastriggers	Boolean	RELHASTRIGGERS in PG_CLASS	Specifies whether the table has triggers. If it does, the value is TRUE . Otherwise, the value is FALSE .
tablecreator	name	pg_get_userbyid(creator in PG_OBJECT)	Table creator.
created	timestamp with time zone	ctime in PG_OBJECT	Time when the table is created.
last_ddl_time	timestamp with time zone	mtime in PG_OBJECT	Time when the last DDL operation is performed on the table.

12.3.152 PG_TDE_INFO

PG_TDE_INFO displays encryption information of the entire cluster.

Table 12-242 PG_TDE_INFO columns

Name	Type	Description
is_encrypt	Boolean	Specifies whether the cluster is an encryption cluster. • f: non-encryption cluster. • t: encryption cluster.
g_tde_algo	text	Encryption algorithm. • SM4-CTR-128 • AES-CTR-128
remain	text	Reserved column.

12.3.153 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays information about all available time zones.

Table 12-243 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 TRUE ; otherwise, the value is FALSE .

12.3.154 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 12-244 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.
is_dst	Boolean	Specifies whether the DST is being used. If the DST is being used, the value is TRUE ; otherwise, the value is FALSE .

12.3.155 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays memory usage of a node in the database.

Table 12-245 PG_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name
memorytype	text	Memory name

Name	Type	Description
memorybytes	integer	Size of the used memory, in MB

12.3.156 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 12-246 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 0 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.
total_space	bigint	Available permanent table storage space, in KB (-1 if the storage space is not limited).
used_temp_space	bigint	Used temporary storage 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).

Name	Type	Description
read_kbytes	bigint	CN: total bytes read by the user's complex jobs on all DN's 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 DN's 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 DN's 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 DN's 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.

12.3.157 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 12-247 PG_TOTAL_USER_RESOURCE_INFO_OID columns

Name	Type	Description
userid	oid	User ID.

Name	Type	Description
used_memory	integer	Used memory, in MB.
total_memory	integer	Available memory, in MB. The value 0 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 storage 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	Amount of data read from the disk, in KB.
write_kbytes	bigint	Amount 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.

12.3.158 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 permissions.

Table 12-248 PG_USER columns

Name	Type	Description
username	name	Username.

Name	Type	Description
usesysid	oid	ID of this user.
usecreatedb	Boolean	Specifies whether a user has the permissions to create databases. • t (true): yes. • f (false): no.
usesuper	Boolean	Specifies whether the user is the initial system administrator with the highest permissions. • t (true): yes. • f (false): no.
usecatupd	Boolean	Specifies whether the user can directly update system catalogs. Only the initial system administrator whose usesysid is 10 has this permission. This permission is unavailable for other users. • t (true): yes. • f (false): no.
userepl	Boolean	Specifies whether the user has the permissions to duplicate data streams. • t (true): yes. • f (false): no.
passwd	text	Encrypted user password. The value is displayed as ***** .
valbegin	timestamp with time zone	Start time for account validity (NULL if start time is not specified).
valuntil	timestamp with time zone	End time for account validity (NULL 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.
useconfig	text[]	Default value of runtime configuration items. For details, see setconfig in PG_DB_ROLE_SETTING .
nodegroup	name	Name of the node group associated with a user. If the user does not manage node groups, leave it blank.

Name	Type	Description
tempspacelimit	text	Storage space of the temporary table, in KB.
spillspacelimit	text	Operator disk spill space, in KB.
usemonitoradmin	Boolean	Specifies whether the user has the MONADMIN permission. • t (true): yes. • f (false): no.
useoperatoradmin	Boolean	Specifies whether the user is an O&M administrator. • t (true): yes. • f (false): no.
usepolicyadmin	Boolean	Specifies whether this user is a POLADMIN. • t (true): yes. • f (false): no.

12.3.159 PG_USER_MAPPINGS

PG_USER_MAPPINGS displays user mapping information. All users can view the report.

Table 12-249 PG_USER_MAPPINGS columns

Name	Type	Reference	Description
umid	oid	oid in PG_USER_MAPPING	OID of the user mapping.
srvid	oid	oid in PG_FOREIGN_SERVER	OID of the foreign server that contains the mapping.
srvname	name	srvname in PG_FOREIGN_SERVER	Name of the foreign server.
umuser	oid	oid in PG_AUTHID	OID of the local role being mapped (0 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 NULL .

12.3.160 PG_VARIABLE_INFO

PGXC_VARIABLE_INFO records information about transaction IDs and OIDs of the current node in a cluster.

Table 12-250 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 datafrozensxid 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_comple ted_xid	xid	Latest transaction ID on the node after the transaction commit or rollback.
startup_max_xi d	xid	Last transaction ID before the node is powered off.

12.3.161 PG_VIEWS

PG_VIEWS displays useful information about each view in the database.

Table 12-251 PG_VIEWS columns

Name	Type	Reference	Description
schemaname	name	nspname in PG_NAMESPACE	Schema name of a view.
viewname	name	relname in PG_CLASS	View name.
viewowner	name	Erolname in PG_AUTHID	Owner of the view.
definition	text	-	Definition of the view.

12.3.162 PGXC_COMM_DELAY

PGXC_COMM_DELAY displays the communications library delay status of all DNs. Only SYSADMIN and MONADMIN can view the status.

Table 12-252 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 NOTE 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

12.3.163 PGXC_COMM_RECV_STREAM

PGXC_COMM_RECV_STREAM displays the receiving stream status of the communication libraries for all the DNs.

Table 12-253 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 • UNKNOWN: The logical connection status is unknown. • READY: The logical connection is ready. • RUN: The logical connection sends packets normally. • HOLD: The logical connection is waiting to send packets. • CLOSED: The logical connection is closed. • TO_CLOSED: The logical connection will be closed.
query_id	bigint	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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

12.3.164 PGXC_COMM_SEND_STREAM

PGXC_COMM_SEND_STREAM displays the sending stream status of the communication libraries for all the DNs.

Table 12-254 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: ● UNKNOWN: The connection status is unknown. ● READY: The connection is ready. ● RUN: The connection sends packets normally. ● HOLD: The connection is waiting to send packets. ● CLOSED: The connection is closed. ● TO_CLOSED: The connection will be closed.
query_id	bigint	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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

12.3.165 PGXC_COMM_STATUS

PGXC_COMM_STATUS displays the communications library status for all the DN's.

Table 12-255 PGXC_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 KB/s
txkbyte_rate	bigint	Sending rate of the communications library on the node, in KB/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the libcomm process, in bytes
memkbyte_libpq	bigint	Communication memory size of the libpq process, in bytes
used_pm	integer	Real-time usage of the postmaster thread
used_sflow	integer	Real-time usage of the gs_sender_flow_controller thread
used_rflow	integer	Real-time usage of the gs_receiver_flow_controller thread
used_rloop	integer	Highest real-time usage among multiple gs_receivers_loop threads
stream	integer	Total number of used logical connections.

12.3.166 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 SYSADMIN or MONADMIN permission can view the information.

Table 12-256 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

12.3.167 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS displays the data skew on tables in the current database. Only users with the SYSADMIN permission can access this view.

Table 12-257 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.)

12.3.168 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 SYSADMIN or MONADMIN permission.

Table 12-258 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.

12.3.169 PGXC_OS_THREADS

PGXC_OS_THREADS displays thread status information under all normal nodes in the current cluster. Only users with the SYSADMIN and MONADMIN permissions can query this view.

Table 12-259 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 pid .
thread_name	text	Name of the thread corresponding to pid .
creation_time	timestamp with time zone	Creation time of the thread corresponding to pid .

12.3.170 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays two-phase transactions in the prepared phase. Only users with the SYSADMIN or MONADMIN permission can view the information.

Table 12-260 PGXC_PREPARED_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Displays the two-phase transaction in the prepared phase.

12.3.171 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 SYSADMIN or MONADMIN permission can view the information.

Table 12-261 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. • 3: prepared. • 0: starting.
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is lazy vacuum. • t (true): yes. • f (false): no.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the prepared state. If the status is not prepared , the value is 0 .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent from a CN to a DN.

12.3.172 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 SYSADMIN or MONADMIN 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 12-159](#).

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.

12.3.173 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 12-262 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 null , 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 client_addr . This column will be non-null only for IP connections and only when log_hostname 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 the 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 query_start column.

Name	Type	Description
query_start	timestamp with time zone	Time when the currently active query was started, or if state is not active , when the last query was started.
state_change	timestamp with time zone	Time when state 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 true . Otherwise, the value is false .
enqueue	text	Queuing status of a statement. Its value can be: • waiting in queue: The statement is in the queue. • Empty: The statement is running.

Name	Type	Description
state	text	Overall status of this backend. Its value can be: • active: The backend is executing a query. • idle: The backend is waiting for a new client command. • idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction. • idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction. • fastpath function call: The backend is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in this backend. NOTE Only users with the SYSADMIN permission can view the session status of their accounts. The state information of other accounts is empty. For example, after user judy is connected to the database, the state information of user joe and the initial user omm in pgxc_stat_activity is empty. <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 state is active , 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.
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.

12.3.174 PGXC_STAT_BAD_BLOCK

PGXC_STAT_BAD_BLOCK shows statistics about Page or CU verification failures after all nodes in a cluster are started. This view can be viewed only by the SYSADMIN and MONADMIN users.

Table 12-263 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 The values are as follows: ● 0: main data file. ● 1: FSM file. ● 2: VM file. ● 3: 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

12.3.175 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 12-264 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 debug_query_id .
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.
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.

```
openGauss=# SELECT * FROM pg_thread_wait_status WHERE query_id > 0;
 node_name | db_name | thread_name | query_id |   tid   | lwtid | ptid | tlevel | smpid |
 wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 coordinator1 | postgres | gsql      | 20971544 | 140274089064208 | 22579 |    |    | 0 | 0 | wait node:
```

```
datanode4 |
(1 rows)
openGauss=# SELECT * FROM pgxc_thread_wait_status WHERE query_id > 0;
 node_name | db_name | thread_name | query_id | tid | sessionid | lwtid | psessionid |
 tlevel | smpid | wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 coordinator1 | postgres | gsql | 77687093572155050 | 47212704827136 | 47212704827136 | 63191
 | 0 | 0 | none |
 coordinator2 | postgres | coordinator1 | 77687093572155050 | 47403117319936 | 47403117319936 |
 159322 | 0 | 0 | none |
 data_node1 | postgres | coordinator1 | 77687093572155050 | 47723869374208 | 47723869374208 |
 159320 | 0 | 0 | none |
 data_node2 | postgres | coordinator1 | 77687093572155050 | 47852867290880 | 47852867290880 |
 159321 | 0 | 0 | none |
(4 rows)
```

12.3.176 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 or MONADMIN permission can query this view.

Table 12-265 PGXC_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory name. • max_process_memory: memory occupied by a GaussDB cluster instance. • process_used_memory: memory occupied by a GaussDB process. • max_dynamic_memory: maximum dynamic memory. • dynamic_used_memory: used dynamic memory. • dynamic_peak_memory: dynamic memory peak. • dynamic_used_shrctx: maximum dynamic shared memory context. • dynamic_peak_shrctx: dynamic peak value of the shared memory context. • max_shared_memory: maximum shared memory. • shared_used_memory: used shared memory. • max_sctpcomm_memory: maximum memory allowed for the communications library. • sctpcomm_used_memory: memory used by the communications library. • sctpcomm_peak_memory: memory peak of the communications library. • other_used_memory: other used memory.
memorybytes	integer	Size of the used memory (unit: MB).

12.3.177 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 SYSADMIN and MONADMIN 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 12-266 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 datafrozensxid 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_completd_xid	xid	Latest transaction ID on the node after the transaction commit or rollback
startup_max_xid	xid	Last transaction ID before the node is powered off

12.3.178 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.

Table 12-267 PLAN_TABLE columns

Name	Type	Description
statement_id	character varying(30)	Query tag specified by a user.
plan_id	bigint	Query ID.
id	int	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	float8	Execution cost estimated by the optimizer for an operator.
cardinality	float8	Number of rows estimated by the optimizer for an operator.

 NOTE

- A valid **object_type** value consists of a relkind type defined in **PG_CLASS** (**TABLE**, **INDEX**, **SEQUENCE**, **VIEW**, **COMPOSITE TYPE**, 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.

12.3.179 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 12-268 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

12.3.180 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 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: time spent on I/Os.

Table 12-269 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 μ s.

12.3.181 PV_OS_RUN_INFO

PV_OS_RUN_INFO displays the running status of the OS.

Table 12-270 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

12.3.182 PV_REDO_STAT

PV_REDO_STAT displays the log replay of session threads.

Table 12-271 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.

12.3.183 PV_SESSION_MEMORY

PV_SESSION_MEMORY displays the 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 12-272 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.

12.3.184 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 12-273 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 TempSmallContextGroup , this parameter specifies the number of collected memory contexts.

12.3.185 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}/dumpmem* directory. *threadid* can be obtained from **sessid** in the following table.

Table 12-274 PV_SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	1. When the thread pool is disabled (enable_thread_pool is set to off), this column indicates the thread start time +session ID (string: <i>timestamp.sessionid</i>). 2. When the thread pool is enabled (enable_thread_pool is set to on): 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>).
sesstype	text	Thread name.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.

Name	Type	Description
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 TempSmallContextGroup , this parameter specifies the number of collected memory contexts.

12.3.186 PV_SESSION_STAT

PV_SESSION_STAT collects statistics about session states based on session threads or the AutoVacuum thread.

Table 12-275 PV_SESSION_STAT columns

Name	Type	Description
sessid	text	Thread ID and start time.
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.

12.3.187 PV_SESSION_TIME

PV_SESSION_TIME displays the running time of session threads and time consumed in each execution phase.

Table 12-276 PV_SESSION_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time.
stat_id	integer	Statistics ID.

Name	Type	Description
stat_name	text	Name of the session type. • 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.
value	bigint	Session value.

12.3.188 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 12-277 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.

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 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 TempSmallContextGroup , this parameter specifies the number of collected memory contexts.

12.3.189 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

Table 12-278 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: • max_process_memory: memory occupied by a GaussDB cluster instance. • process_used_memory: memory occupied by a GaussDB process. • max_dynamic_memory: maximum dynamic memory. • dynamic_used_memory: used dynamic memory. • dynamic_peak_memory: dynamic memory peak. • dynamic_used_shrctx: maximum dynamic shared memory context. • dynamic_peak_shrctx: dynamic peak value of the shared memory context. • max_backend_memory: maximum memory that can be used when the HA port is used to execute services. • backend_used_memory: memory that has been used when the HA port is used to execute services. • max_shared_memory: maximum shared memory. • shared_used_memory: used shared memory. • max_sctpcomm_memory: maximum memory allowed for the communications library. • sctpcomm_used_memory: memory used by the communications library. • sctpcomm_peak_memory: memory peak of the communications library. • other_used_memory: other used memory.
memorybytes	integer	Size of allocated memory-typed memory.

12.3.190 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 12-279 SYS_DUMMY columns

Name	Type	Description
dummy	text	Expression calculation result

13 Schemas

The following table describes the schemas supported in GaussDB.

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 users with the MONADMIN permission can view views and functions in this schema by default.
snapshot	Manages data related to WDR snapshots. By default, the initial user or users with the MONADMIN permission can access the data.
sqladvsior	Recommends distribution keys. For details, see Distribution Key Recommendation Functions .
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 search_path 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.
pkg_service	Manages information about the package service.
pkg_util	Manages information about the package tool.
dbe_raw	Advanced function package dbe_raw, 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.

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.

Schema	Description
db4ai	Manages data of different versions in AI training.
dbe_pldevel oper	Compiles and debugs user stored procedures.
dbe_sql_util	Manages statement patches.

The following APIs are not supported in distributed deployment mode:

- `bool dbe_sql_util.create_hint_sql_patch(name, bigint, text, text DEFAULT NULL::text, boolean DEFAULT true)`
- `bool dbe_sql_util.create_abort_sql_patch(name, bigint, text DEFAULT NULL::text, boolean DEFAULT true)`
- `bool dbe_sql_util.drop_sql_patch(name)`
- `bool dbe_sql_util.enable_sql_patch(name)`
- `bool dbe_sql_util.disable_sql_patch(name)`
- `record dbe_sql_util.show_sql_patch(patch_name name, OUT unique_sql_id bigint, OUT enable boolean, OUT abort boolean, OUT hint_str text)`

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 (0 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 defined in.

foreign_server_name	information_schema.s ql_identifier	Name of the foreign server.
srvowner	information_schema.s ql_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 users with the MONADMIN permission can view views and functions in the **DBE_PERF** scheme by default. If the database is upgraded from an earlier version, permissions for the **DBE_PERF** schema are the same as those of the earlier version to ensure forward compatibility. 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	Type	Description
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 entire 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 pid
thread_name	text	Name of the thread corresponding to pid

Name	Type	Description
creation_time	timestamp with time zone	Creation time of the thread corresponding to pid

13.2.1.4 GLOBAL_OS_THREADS

GLOBAL_OS_THREADS records the thread status information on all normal nodes in the entire 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 pid
thread_name	text	Name of the thread corresponding to pid
creation_time	timestamp with time zone	Creation time of the thread corresponding to pid

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 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 13-13 INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID.
stat_name	text	Type name.
value	bigint	Time value (unit: μ s).

13.2.2.2 GLOBAL_INSTANCE_TIME

GLOBAL_INSTANCE_TIME records the time consumption information on all normal nodes in the entire 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: μ s)

13.2.3 Memory

13.2.3.1 MEMORY_NODE_DETAIL

Displays memory usage of a node in the database.

Table 13-15 MEMORY_NODE_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory name. • max_process_memory: memory occupied by a GaussDB instance. • process_used_memory: memory occupied by a process. • max_dynamic_memory: maximum dynamic memory. • dynamic_used_memory: used dynamic memory. • dynamic_peak_memory: dynamic memory peak. • dynamic_used_shrctx: maximum dynamic shared memory context. • dynamic_peak_shrctx: dynamic peak value of the shared memory context. • max_shared_memory: maximum shared memory. • shared_used_memory: used shared memory. • max_sctpcomm_memory: maximum memory allowed for TCP proxy communications. • sctpcomm_used_memory: used memory for TCP proxy communications. • sctpcomm_peak_memory: memory peak of TCP proxy communications. • other_used_memory: other used memory. • gpu_max_dynamic_memory: maximum dynamic GPU memory. • gpu_dynamic_used_memory: used dynamic GPU memory. • gpu_dynamic_peak_memory: dynamic GPU memory peak. • pooler_conn_memory: memory allocated to the connection pool. • pooler_freeconn_memory: memory occupied by idle connections in the connection pool. • storage_compress_memory: memory used by the storage module for compression. • udf_reserved_memory: reserved memory for the UDF.
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	Memory name. • max_process_memory: maximum available memory of the database node. • process_used_memory: memory occupied by a process. • max_dynamic_memory: maximum dynamic memory. • dynamic_used_memory: used dynamic memory. • dynamic_peak_memory: dynamic memory peak. • dynamic_used_shrctx: context of the used dynamic shared memory. • dynamic_peak_shrctx: dynamic peak value of the shared memory context. • max_shared_memory: maximum shared memory. • shared_used_memory: used shared memory. • max_sctpcomm_memory: maximum memory allowed for TCP proxy communications. • sctpcomm_used_memory: used memory for TCP proxy communications. • sctpcomm_peak_memory: memory peak of TCP proxy communications. • other_used_memory: other used memory. • gpu_max_dynamic_memory: maximum dynamic GPU memory. • gpu_dynamic_used_memory: used dynamic GPU memory. • gpu_dynamic_peak_memory: dynamic GPU memory peak. • pooler_conn_memory: memory allocated to the connection pool. • pooler_freeconn_memory: memory occupied by idle connections in the connection pool. • storage_compress_memory: memory used by the storage module for compression. • udf_reserved_memory: reserved memory for the user-defined functions.
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 • ng_total_memory: total memory configured in the node group • ng_used_memory: used memory • ng_estimate_memory: memory used by the optimizer for evaluation • ng_foreignrp_memsize: memory configured in the foreign resource pool • ng_foreignrp_usedsize: memory used by the foreign resource pool • ng_foreignrp_peaksize: peak memory used by the foreign resource pool • ng_foreignrp_mempct: percentage of the system memory configured in attributes of the foreign resource pool • ng_foreignrp_estmsize: memory used by the optimizer for job evaluation in the foreign resource pool
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 entire 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 allocation 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 function.

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 allocation location belongs.

Name	Type	Description
line	integer	Line number of the memory allocation 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: μ s)
writetim	bigint	Total duration of writing (unit: μ s)
avgiotim	bigint	Average duration of reading and writing (unit: μ s)
lstiotim	bigint	Duration of the last file reading (unit: μ s)
miniotim	bigint	Minimum duration of reading and writing (unit: μ s)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μ s)

13.2.4.2 SUMMARY_FILE_IOSTAT

Records statistics about data file I/Os in the cluster to reflect performance issues such as exceptions in I/O operations.

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: μ s).
writetim	numeric	Total duration of writing (unit: μ s).
avgiotim	bigint	Average duration of reading and writing (unit: μ s).
lstiotim	bigint	Duration of the last file reading (unit: μ s).
miniotim	bigint	Minimum duration of reading and writing (unit: μ s).
maxiowtm	bigint	Maximum duration of reading and writing (unit: μ s).

13.2.4.3 GLOBAL_FILE_IOSTAT

Displays statistics about data file I/Os on all nodes.

Table 13-23 GLOBAL_FILE_IOSTAT columns

Name	Type	Description
node_name	name	Node name.
filenum	oid	File ID.

Name	Type	Description
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: μ s).
writetim	bigint	Total duration of writing (unit: μ s).
avgiotim	bigint	Average duration of reading and writing (unit: μ s).
lstiotim	bigint	Duration of the last file reading (unit: μ s).
miniotim	bigint	Minimum duration of reading and writing (unit: μ s).
maxiowtm	bigint	Maximum duration of reading and writing (unit: μ 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: μ s).
avgiotim	bigint	Average duration of writing into Xlog files (unit: μ s). The value is writetim divided by phywrts .

Name	Type	Description
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μ s).

13.2.4.5 SUMMARY_FILE_REDO_IOSTAT

Displays statistics about redo logs (WALs) in the cluster.

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.
writetim	numeric	Duration of writing into Xlog files (unit: μ s).
avgiotim	bigint	Average duration of writing into Xlog files (unit: μ s). The value is writetim divided by phywrts .
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μ 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.

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: μ s).
avgiotim	bigint	Average duration of writing into Xlog files (unit: μ s). The value is writetim divided by phywrts .
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μ s).
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μ s).
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μ 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

Displays statistics about data file I/Os on 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

STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces 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 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 dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.2 SUMMARY_STAT_USER_TABLES

SUMMARY_STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces in the cluster.

Table 13-31 SUMMARY_STAT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
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 HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed

Name	Type	Description
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.3 GLOBAL_STAT_USER_TABLES

GLOBAL_STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces on each node.

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 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.

Name	Type	Description
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	Last time at which this table was manually vacuumed (excluding VACUUM FULL).
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
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 this table has been analyzed by the autovacuum daemon

13.2.5.4 STAT_USER_INDEXES

STAT_USER_INDEXES displays the index status information about user-defined ordinary tables in the current database.

Table 13-33 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

Name	Type	Description
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 table rows fetched by simple index scans using the index

13.2.5.5 SUMMARY_STAT_USER_INDEXES

SUMMARY_STAT_USER_INDEXES displays the index status information about user-defined ordinary tables in all databases in the cluster.

Table 13-34 SUMMARY_STAT_USER_INDEXES columns

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	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

13.2.5.6 GLOBAL_STAT_USER_INDEXES

GLOBAL_STAT_USER_INDEXES displays the index status information about user-defined ordinary tables on each node.

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

Name	Type	Description
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 table rows fetched by simple index scans using the index

13.2.5.7 STAT_SYS_TABLES

STAT_SYS_TABLES displays statistics about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on a single node.

Table 13-36 STAT_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
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)

Name	Type	Description
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 (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.8 SUMMARY_STAT_SYS_TABLES

SUMMARY_STAT_SYS_INDEXES displays statistics about all system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas in the cluster.

Table 13-37 SUMMARY_STAT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
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 HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.9 GLOBAL_STAT_SYS_TABLES

GLOBAL_STAT_SYS_TABLES displays statistics about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on each node in the cluster.

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 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 dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon

Name	Type	Description
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.10 STAT_SYS_INDEXES

STAT_SYS_INDEXES displays the index status information about all system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas.

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 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 table rows fetched by simple index scans using the index

13.2.5.11 SUMMARY_STAT_SYS_INDEXES

SUMMARY_STAT_SYS_INDEXES displays the index status information about all system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas in the cluster.

Table 13-40 SUMMARY_STAT_SYS_INDEXES columns

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	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

13.2.5.12 GLOBAL_STAT_SYS_INDEXES

GLOBAL_STAT_SYS_INDEXES displays the index status information about all system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on each node.

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
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 table rows fetched by simple index scans using the index

13.2.5.13 STAT_ALL_TABLES

STAT_ALL_TABLES displays statistics about one row for each table (including TOAST tables) in databases on the current node.

Table 13-42 STAT_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
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 dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.14 SUMMARY_STAT_ALL_TABLES

Displays statistics about a row of each table (including TOAST tables) in the cluster.

Table 13-43 SUMMARY_STAT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
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 HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed

Name	Type	Description
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.15 GLOBAL_STAT_ALL_TABLES

GLOBAL_STAT_ALL_TABLES displays statistics about a row of each table (including TOAST tables) on each node.

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 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)

Name	Type	Description
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 (not counting VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
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 this table has been analyzed by the autovacuum daemon

13.2.5.16 STAT_ALL_INDEXES

STAT_ALL_INDEXES displays the access information of each index on the current 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 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

Name	Type	Description
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

13.2.5.17 SUMMARY_STAT_ALL_INDEXES

SUMMARY_STAT_ALL_INDEXES displays the access information of each index in the cluster.

Table 13-46 SUMMARY_STAT_ALL_INDEXES columns

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	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

13.2.5.18 GLOBAL_STAT_ALL_INDEXES

GLOBAL_STAT_ALL_INDEXES displays the access information of each index on each node in the cluster.

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 that the index is in
relname	name	Name of the table for the index.

Name	Type	Description
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_fetched	bigint	Number of live table rows fetched by simple index scans using the index

13.2.5.19 STAT_DATABASE

Contains statistics about each database on the current node.

Table 13-48 STAT_DATABASE columns

Name	Type	Description
datid	oid	Database OID.
datname	name	Database name.
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
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 times disk blocks were found in the cache (that is, the buffer cache, not the file system cache of the OS), so that a read is not necessary
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

Name	Type	Description
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 STAT_DATABASE_CONFLICTS .
temp_files	bigint	Number of temporary files created by queries in this 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.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting.
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.

13.2.5.20 SUMMARY_STAT_DATABASE

Contains statistics about each database in the cluster.

Table 13-49 SUMMARY_STAT_DATABASE columns

Name	Type	Description
datname	name	Database name.
numbackends	bigint	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.

Name	Type	Description
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 times disk blocks were found in the cache (that is, the buffer cache, not the file system cache of the OS), so that a read is not necessary
tup_returned	numeric	Number of rows returned by queries in this database
tup_fetched	numeric	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 STAT_DATABASE_CONFLICTS .
temp_files	numeric	Number of temporary files created by queries in this 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.
temp_bytes	numeric	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting.
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.

13.2.5.21 GLOBAL_STAT_DATABASE

GLOBAL_STAT_DATABASE contains statistics about databases on each node in the cluster.

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. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	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 times disk blocks were found in the buffer cache (unnecessary as the number includes only hits in the database kernel buffer cache)
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

Name	Type	Description
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 STAT_DATABASE_CONFLICTS .
temp_files	bigint	Number of temporary files created by queries in this 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.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting.
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.

13.2.5.22 STAT_DATABASE_CONFLICTS

STAT_DATABASE_CONFLICTS displays statistics about database conflicts on the current 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.

Name	Type	Description
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

SUMMARY_STAT_DATABASE_CONFLICTS displays statistics about database conflicts in the cluster.

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

GLOBAL_STAT_DATABASE_CONFLICTS displays statistics about database conflicts on each node.

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.

Name	Type	Description
confl_snapsho t	bigint	Number of conflicting snapshots.
confl_bufferpi n	bigint	Number of conflicting buffers.
confl_deadloc k	bigint	Number of conflicting deadlocks.

13.2.5.25 STAT_XACT_ALL_TABLES

STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in the current namespace.

Table 13-54 STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemanam e	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_rea d	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetc h	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)

13.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES

SUMMARY_STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in namespaces in the cluster.

Table 13-55 SUMMARY_STAT_XACT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
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 HOT updated (with no separate index update required)

13.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES

GLOBAL_STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in namespaces on each node.

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 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.

Name	Type	Description
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

13.2.5.28 STAT_XACT_SYS_TABLES

STAT_XACT_SYS_TABLES displays the transaction status information about the system catalogs in namespaces on the current node.

Table 13-57 STAT_XACT_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.
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)

13.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES

SUMMARY_STAT_XACT_SYS_TABLES displays the transaction status information about the system catalogs in namespaces in the cluster.

Table 13-58 SUMMARY_STAT_XACT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in

Name	Type	Description
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 HOT updated (with no separate index update required)

13.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES

GLOBAL_STAT_XACT_SYS_TABLES displays the transaction status information about the system catalogs in namespaces on each node.

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 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)

13.2.5.31 STAT_XACT_USER_TABLES

STAT_XACT_USER_TABLES displays the transaction status information about the user tables in namespaces 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 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_fetched	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)

13.2.5.32 SUMMARY_STAT_XACT_USER_TABLES

SUMMARY_STAT_XACT_USER_TABLES displays the transaction status information about the user tables in namespaces in the database.

Table 13-61 SUMMARY_STAT_XACT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name.

Name	Type	Description
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 HOT updated (with no separate index update required)

13.2.5.33 GLOBAL_STAT_XACT_USER_TABLES

GLOBAL_STAT_XACT_USER_TABLES displays the transaction status information about the user tables in namespaces on each node.

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 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)

13.2.5.34 STAT_XACT_USER_FUNCTIONS

STAT_XACT_USER_FUNCTIONS displays statistics about function executions in the current transaction.

Table 13-63 STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function ID.
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 this function and all other functions called by it
self_time	double precision	Total time spent in this function, excluding other functions called by it

13.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

SUMMARY_STAT_XACT_USER_FUNCTIONS displays statistics about function executions in the current transaction in the cluster.

Table 13-64 SUMMARY_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name
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	Total time spent in this function, excluding other functions called by it

13.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

GLOBAL_STAT_XACT_USER_FUNCTIONS displays statistics about function executions in transactions on each node.

Table 13-65 GLOBAL_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.
funcid	oid	Function ID.
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 this function and all other functions called by it.
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.
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

Displays the information about table and index read failures in the cluster.

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.
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

Displays information about table and index read failures on each node.

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 occurred for the last time.

13.2.5.40 STAT_USER_FUNCTIONS

STAT_USER_FUNCTIONS displays the status information about user-defined functions in the current namespace. (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	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 (unit: ms)
self_time	double precision	Total time spent in this function, excluding other functions called by it (unit: ms)

13.2.5.41 SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS collects statistics about user-defined views in the cluster.

Table 13-70 SUMMARY_STAT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name
funcname	name	Function name
calls	numeric	Number of times that the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it (unit: ms)
self_time	double precision	Total 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.

Table 13-71 GLOBAL_STAT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.

Name	Type	Description
funcid	oid	ID of the function.
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 this function and all 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. Common users can view only the distribution of SQL statements executed by themselves in workloads, whereas users with the MONADMIN permission can view the overall load status of workloads.

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 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

13.2.6.3 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: μ s)
resp_max	bigint	Maximum response time of user transactions (unit: μ s)
resp_avg	bigint	Average response time of user transactions (unit: μ s)

Name	Type	Description
resp_total	bigint	Total response time of user transactions (unit: μ 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: μ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: μ s)
bg_resp_total	bigint	Total response time of background transactions (unit: μ 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: μ s)
resp_max	bigint	Maximum response time of user transactions (unit: μ s)
resp_avg	bigint	Average response time of user transactions (unit: μ s)
resp_total	numeric	Total response time of user transactions (unit: μ s)
bg_commit_counter	numeric	Number of background transactions committed
bg_rollback_counter	numeric	Number of background transactions rolled back

Name	Type	Description
bg_resp_min	bigint	Minimum response time of background transactions (unit: μ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: μ s)
bg_resp_total	numeric	Total response time of background transactions (unit: μ 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: μ s)
resp_max	bigint	Maximum response time of user transactions (unit: μ s)
resp_avg	bigint	Average response time of user transactions (unit: μ s)
resp_total	bigint	Total response time of user transactions (unit: μ 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: μ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μ s)

Name	Type	Description
bg_resp_avg	bigint	Average response time of background transactions (unit: μ s)
bg_resp_total	bigint	Total response time of background transactions (unit: μ s)

13.2.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME collects statistics about SUIDs in workloads.

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: μ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μ s).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μ s).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s).

Name	Type	Description
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s).

13.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME collects statistics about SUIDs in service workloads 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: μ s).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μ s).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μ s).

Name	Type	Description
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μ s).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ s).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s).

13.2.6.8 USER_TRANSACTION

USER_TRANSACTION collects statistics about transactions executed by users. Common users can view only transactions executed by themselves, whereas users with the MONADMIN permission can view 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: μ s)
resp_max	bigint	Maximum response time of user transactions (unit: μ s)
resp_avg	bigint	Average response time of user transactions (unit: μ s)
resp_total	bigint	Total response time of user transactions (unit: μ 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: μ s)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: μ s)
bg_resp_total	bigint	Total response time of background transactions (unit: μ 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: μ s)
resp_max	bigint	Maximum response time of user transactions (unit: μ s)
resp_avg	bigint	Average response time of user transactions (unit: μ s)
resp_total	bigint	Total response time of user transactions (unit: μ 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: μ s)

Name	Type	Description
bg_resp_max	bigint	Maximum response time of background transactions (unit: μ s)
bg_resp_avg	bigint	Average response time of background transactions (unit: μ s)
bg_resp_total	bigint	Total response time of background transactions (unit: μ 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

Name	Type	Description
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

Collects statistics about the running time of session threads and time consumed in each execution phase on each node, as described in [Table 13-84](#).

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 the current node for jobs currently executed by users, as described in [Table 13-85](#).

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 DN for jobs currently executed by users, as described in [Table 13-86](#).

Table 13-86 GLOBAL_SESSION_MEMORY columns

Name	Type	Description
node_name	name	Node name.
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.

13.2.7.7 SESSION_MEMORY_DETAIL

Collects statistics on the memory usage of threads on the current node, in the unit specified by **MemoryContext**, as described in [Table 13-87](#).

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

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)

13.2.7.8 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
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 THREAD_WAIT_STATUS

This view allows you to test the blocking status of the backend thread and auxiliary thread on the current node.

Table 13-89 THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Current node name.
db_name	text	Database name.
thread_name	text	Thread name.

Name	Type	Description
query_id	bigint	Query ID. The value of this column is the same as that of debug_query_id .
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.
wait_event	text	If wait_status is acquire lock , acquire lwlock , or wait io , 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.
global_sessionid	text	Global session ID.

13.2.7.10 GLOBAL_THREAD_WAIT_STATUS

In this view, you can check the blocking and waiting status of backend threads and auxiliary threads on all nodes.

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 13-90 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 debug_query_id .
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.
wait_event	text	If wait_status is acquire lock , acquire lwlock , or wait io , 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.
global_sessionid	text	Global session ID.

13.2.7.11 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 13-91 LOCAL_THREADPOOL_STATUS columns

Name	Type	Description
node_name	text	Node name.

Name	Type	Description
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 NULL .
listener	integer	Number of listener threads in the thread pool group.
worker_info	text	Information about threads in the thread pool, including: ● default: Number of initial threads in the thread pool group ● new: Number of new threads in the thread pool group ● expect: Expected number of threads in the thread pool group ● actual: Actual number of threads in the thread pool group ● idle: Number of idle threads in the thread pool group ● pending: Number of pending threads in the thread pool group
session_info	text	Information about sessions in the thread pool, including: ● total: Total number of sessions in the thread pool group. ● waiting: Number of sessions pending scheduling in the thread pool group. ● running: Number of running sessions in the thread pool group. ● idle: Number of idle sessions in the thread pool group.
stream_info	text	Information about streams in the thread pool, including: ● total: total number of stream threads in the thread pool group. ● running: number of running stream threads in the thread pool group. ● idle: number of idle stream threads in the thread pool group.

13.2.7.12 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 13-91](#).

13.2.7.13 SESSION_CPU_RUNTIME

SESSION_CPU_RUNTIME displays information about CPU usage of ongoing complex jobs executed by the current user.

Table 13-92 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.
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.
node_group	text	Node group of the user to which the statement belongs.
top_cpu_dn	text	Top <i>N</i> CPU usage.

13.2.7.14 SESSION_MEMORY_RUNTIME

SESSION_MEMORY_RUNTIME displays information about memory usage of ongoing complex jobs executed by the current user.

Table 13-93 SESSION_MEMORY_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that this backend is connected to.

Name	Type	Description
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, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
min_peak_memory	integer	Minimum peak memory of the statement among all DN's (unit: MB).
max_peak_memory	integer	Maximum peak memory of the statement among all DN's (unit: MB).
spill_info	text	Statement spill information on all DN's. • None: No data is spilled to disks on all DN's. • All: Data is spilled to disks on all DN's. • [<i>a</i>:<i>b</i>]: The statement has been spilled to disks on <i>a</i> of <i>b</i> DN's.
query	text	Statement being executed.
node_group	text	Node group of the user to which the statement belongs.
top_mem_dn	text	Top N memory usage.

13.2.7.15 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION displays samples in the **ACTIVE SESSION PROFILE** memory on the current node.

Table 13-94 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.

Name	Type	Description
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.
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 locktag_decode .
lockmode	text	Mode of a lock that the session waits for.

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.
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 active , idle in transaction , fastpath function call , idle in transaction (aborted) , disabled , or retrying .

13.2.7.16 GLOBAL_ACTIVE_SESSION

GLOBAL_ACTIVE_SESSION displays a summary of samples in the **ACTIVE SESSION PROFILE** memory on all nodes.

Table 13-95 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.

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 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 locktag_decode .
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.

Name	Type	Description
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 active , idle in transaction , fastpath function call , idle in transaction (aborted) , disabled , or retrying .

13.2.8 Transaction

13.2.8.1 TRANSACTIONS_RUNNING_XACTS

Displays information about running transactions on the current node.

Table 13-96 TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM.
gxid	xid	Transaction ID.
state	tinyint	Transaction status (3 : prepared; 0 : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • true: yes. • false: no.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the prepared state. If the state is not prepared , the value is 0 .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent from a CN to a DN.

13.2.8.2 SUMMARY_TRANSACTIONS_RUNNING_XACTS

Displays the summary of running transactions on each CN in the cluster.

Table 13-97 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 (3 : prepared; 0 : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	Boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • true: yes. • false: no.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the prepared state. If the state is not prepared , the value is 0 .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent from a CN to a DN.

13.2.8.3 GLOBAL_TRANSACTIONS_RUNNING_XACTS

Displays information about the running transactions on each CN and DN in the cluster.

Table 13-98 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 (3 : prepared; 0 : starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.

Name	Type	Description
vacuum	Boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • true: yes. • false: no.
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the prepared state. If the state is not prepared , the value is 0 .
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent from a CN to a DN.

13.2.8.4 TRANSACTIONS_PREPARED_XACTS

TRANSACTIONS_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 13-99 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 who executes the transaction
database	name	Name of the database in which the transaction is executed

13.2.8.5 SUMMARY_TRANSACTIONS_PREPARED_XACTS

Summarizes the information about the transactions that are ready for two-phase commit on each CN in the cluster.

Table 13-100 SUMMARY_TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction

Name	Type	Description
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 who executes the transaction
database	name	Name of the database in which the transaction is executed

13.2.8.6 GLOBAL_TRANSACTIONS_PREPARED_XACTS

Summarizes the information about the transactions that are ready for two-phase commit on each CN and DN in the cluster.

Table 13-101 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 who 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. To query a view, you must have the SYSADMIN or MONADMIN permission. 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.

Table 13-102 STATEMENT columns

Name	Type	Description
node_name	name	Node name.
node_id	integer	Node ID (node_id in pgxc_node).
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 track_activity_query_size .
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μ s).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μ s).
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μ 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 logical reads.
n_blocks_hit	bigint	Number of memory hits.

Name	Type	Description
n_soft_parse	bigint	Number of soft parsing times. The value of n_soft_parse plus the value of n_hard_parse may be greater than the value of n_calls because the number of subqueries is not counted in the value of n_calls .
n_hard_parse	bigint	Number of hard parsing times. The value of n_soft_parse plus the value of n_hard_parse may be greater than the value of n_calls because the number of subqueries is not counted in the value of n_calls .
db_time	bigint	Valid internal database time, which is accumulated if multiple threads are involved (unit: μ s).
cpu_time	bigint	CPU time (unit: μ s).
execution_time	bigint	Execution time in the executor (unit: μ s).
parse_time	bigint	SQL parsing time (unit: μ s).
plan_time	bigint	SQL plan generation time (unit: μ s).
rewrite_time	bigint	SQL rewriting time (unit: μ s).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μ s).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μ s).
data_io_time	bigint	I/O time (unit: μ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μ 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_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μ 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_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μ 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_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μ 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: μ 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.

Name	Type	Description
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: μ 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).

13.2.9.2 SUMMARY_STATEMENT

Displays all information (including DNs) about executed statements (normalized SQL statements) on each CN.

Table 13-103 SUMMARY_STATEMENT columns

Name	Type	Description
node_name	name	Node name.
node_id	integer	Node ID (node_id in pgxc_node).
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 track_activity_query_size .
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μ s).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μ s).
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μ s).
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 logical reads.
n_blocks_hit	bigint	Number of memory hits.
n_soft_parse	bigint	Number of soft parsing times.
n_hard_parse	bigint	Number of hard parsing times.
db_time	bigint	Valid internal database time, which is accumulated if multiple threads are involved (unit: μ s).
cpu_time	bigint	CPU time (unit: μ s).
execution_time	bigint	Execution time in the executor (unit: μ s).
parse_time	bigint	SQL parsing time (unit: μ s).
plan_time	bigint	SQL plan generation time (unit: μ s).
rewrite_time	bigint	SQL rewriting time (unit: μ s).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μ s).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μ s).
data_io_time	bigint	I/O time (unit: μ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μ 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_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μ 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_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μ 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_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μ 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: μ 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.

Name	Type	Description
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: μ 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).

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

By querying the STATEMENT_COUNT view, a common user can view statistics only about this user on the current node, whereas an administrator can view statistics about 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, if the CN receives a query that contains multiple subqueries, the subqueries are counted separately when they are delivered to a single DN or multiple DNs.

Table 13-104 STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	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

Name	Type	Description
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of SELECT statements (unit: μ s)
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s)
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s)
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s)
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μ s)
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s)
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s)
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s)
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μ s)
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s)
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s)
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s)
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μ s)
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s)
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s)
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ 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 13-105 GLOBAL_STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	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: μ s)
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s)
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s)
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s)
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μ s)
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s)
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s)
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s)
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μ s)
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s)
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s)

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s)
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μ s)
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s)
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s)
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ s)

13.2.9.5 SUMMARY_STATEMENT_COUNT

Displays the summary statistics of five types of statements (SELECT, INSERT, UPDATE, DELETE, and MERGE INTO) and DDL, DML, and DCL statements executed on each database node.

Table 13-106 SUMMARY_STATEMENT_COUNT columns

Name	Type	Description
user_name	text	Username
select_count	numeric	Statistical result of the SELECT statement
update_count	numeric	Statistical result of the UPDATE statement
insert_count	numeric	Statistical result of the INSERT statement
delete_count	numeric	Statistical result of the DELETE statement
mergeinto_count	numeric	Statistical result of the MERGE INTO 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 SELECT statements (unit: μ s)
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μ s)

Name	Type	Description
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μ s)
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μ s)
total_update_elapse	numeric	Total response time of UPDATE statements (unit: μ s)
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μ s)
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μ s)
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μ s)
total_insert_elapse	numeric	Total response time of INSERT statements (unit: μ s)
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μ s)
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μ s)
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μ s)
total_delete_elapse	numeric	Total response time of DELETE statements (unit: μ s)
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μ s)
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μ s)
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μ s)

13.2.9.6 STATEMENT_RESPONSETIME_PERCENTILE

STATEMENT_RESPONSETIME_PERCENTILE obtains the response times of 80% and 95% SQL statements in the cluster.

Table 13-107 STATEMENT_RESPONSETIME_PERCENTILE columns

Name	Type	Description
p80	bigint	Response time of 80% SQL statements in the cluster (unit: μ s)

Name	Type	Description
p95	bigint	Response time of 95% SQL statements in the cluster (unit: μ s)

13.2.9.7 STATEMENT_HISTORY

Displays information about statements executed on the current node. To query a view, you must have the SYSADMIN or MONADMIN permission. The result can be queried only in the system database but cannot be queried in the user database.

Table 13-108 STATEMENT_HISTORY columns

Name	Type	Description
dbname	name	Database name.
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.

Name	Type	Description
n_soft_parse	bigint	Number of soft parsing times. The value of n_soft_parse plus the value of n_hard_parse may be greater than the value of n_calls because the number of subqueries is not counted in the value of n_calls .
n_hard_parse	bigint	Number of hard parsing times. The value of n_soft_parse plus the value of n_hard_parse may be greater than the value of n_calls because the number of subqueries is not counted in the value of n_calls .
query_plan	text	Statement execution plan.
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.
db_time	bigint	Valid database time, which is accumulated if multiple threads are involved (unit: μ s).
cpu_time	bigint	CPU time (unit: μ s).
execution_time	bigint	Execution time in the executor (unit: μ s).
parse_time	bigint	SQL parsing time (unit: μ s).
plan_time	bigint	SQL plan generation time (unit: μ s).
rewrite_time	bigint	SQL rewriting time (unit: μ s).
pl_execution_time	bigint	Execution time of PL/pgSQL statements (unit: μ s).

Name	Type	Description
pl_compilation_time	bigint	Compilation time of PL/pgSQL statements (unit: μ s).
data_io_time	bigint	I/O time (unit: μ s).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μ 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: μ 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_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μ 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: {"time":xxx, "n_calls":xxx, "size":xxx} .

Name	Type	Description
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μ 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}.
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_count	bigint	Number of lightweight lock waits.
lwlock_time	bigint	Time required for lightweight locking (reserved).
lwlock_wait_time	bigint	Time required for lightweight lock waiting.
details	bytea	List of statement lock events, which are recorded in chronological order. The number of records is affected by the track_stmt_details_size parameter. This column is binary and needs to be read using the parsing function <code>pg_catalog.statement_detail_decode</code> . For details, see Other Functions . Events include: Start locking. Complete locking. Start lock waiting. Complete lock waiting. Start unlocking. Complete unlocking. Start lightweight lock waiting. Complete lightweight lock waiting.

Name	Type	Description
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.
advise	text	Risks which may cause slow SQL statements. (Multiple risks may exist at the same time.) • Cast Function Cause Index Miss.: Index matching may fail due to implicit conversion. • Limit too much rows.: The SQL statement execution may slow down due to a large limit value. • Proleakproof of function is false.: The proleakproof of the function is set to false. In this case, the function does not use statistics when generating a plan due to data leakage risks. As a result, the accuracy of the generated plan is affected and the SQL statement execution may slow down.

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 13-109 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

Name	Type	Description
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

13.2.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES displays a summary of I/O status information about user relationship tables of each node in the cluster.

Table 13-110 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 13-111 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 13-112 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

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	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 a summary of I/O status information about all user relationship table indexes in namespaces in the cluster.

Table 13-113 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 13-114 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

Name	Type	Description
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

13.2.10.7 STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE displays I/O status information about all user relationship table sequences in namespaces on the current node.

Table 13-115 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 13-116 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 13-117 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 the I/O status information about all system catalogs in namespaces on the current node.

Table 13-118 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

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

13.2.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES displays a summary of I/O status information about system catalogs of each node in the cluster.

Table 13-119 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 13-120 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
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 13-121 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

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

13.2.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES displays a summary of I/O status information about all system catalog indexes in namespaces in the cluster.

Table 13-122 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 13-123 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

Name	Type	Description
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 13-124 STATIO_SYS_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

13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES displays I/O status information about all system sequences in namespaces in the cluster.

Table 13-125 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 13-126 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 displays the I/O status information about each table (including TOAST tables) of the current node in the database.

Table 13-127 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 displays a summary of the I/O status information about each table (including TOAST tables) of each node in the cluster.

Table 13-128 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
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 13-129 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

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

13.2.10.22 STATIO_ALL_INDEXES

STATIO_ALL_INDEXES contains one row for each index in the current database, showing I/O statistics about specific indexes.

Table 13-130 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 one row for each index in the cluster, showing a summary of I/O statistics about specific indexes.

Table 13-131 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
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 13-132 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 13-133 STATIO_ALL_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

13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES contains I/O statistics about one row of each sequence in databases in the cluster.

Table 13-134 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 13-135 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

13.2.10.28 GLOBAL_STAT_DB_CU

GLOBAL_STAT_DB_CU displays CU hits on each node in each database in the cluster. You can clear it using `gs_stat_reset()`.

Table 13-136 GLOBAL_STAT_DB_CU columns

Name	Type	Description
node_name1	text	Node name
db_name	text	Database name
mem_hit	bigint	Number of memory hits
hdd_sync_read	bigint	Number of hard disk synchronous reads
hdd_asyn_read	bigint	Number of hard disk asynchronous reads

13.2.10.29 GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_SESSION_CU displays CU hits of running sessions on each node in the entire cluster. This data about a session is cleared when you exit this session. After the cluster is restarted, the statistics are also cleared.

Table 13-137 GLOBAL_STAT_SESSION_CU columns

Name	Type	Description
node_name1	text	Node name
mem_hit	integer	Number of memory hits
hdd_sync_read	integer	Number of hard disk synchronous reads
hdd_asyn_read	integer	Number of hard disk asynchronous reads

13.2.11 Communications Library

13.2.11.1 COMM_DELAY

COMM_DELAY displays the TCP proxy communications library status for a single DN.

Table 13-138 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: μ s) NOTE 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: μ s)
max_delay	integer	Maximum delay of the current physical connection within 1 minute (unit: μ s)

13.2.11.2 GLOBAL_COMM_DELAY

GLOBAL_COMM_DELAY displays the TCP proxy communications library status for all the DNs.

Table 13-139 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: μ s) NOTE 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: μ s)

Name	Type	Description
max_delay	integer	Maximum delay of the current physical connection within 1 minute (unit: μ 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 13-140 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	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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 DN.

Table 13-141 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	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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 13-142 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
state	text	Stream status
query_id	bigint	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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.

Table 13-143 GLOBAL_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	debug_query_id corresponding to the stream
pn_id	integer	plan_node_id of the query executed by the stream
send_smp	integer	smpid of the sender of the query executed by the stream
recv_smp	integer	smpid 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 13-144 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

Name	Type	Description
rxkbyte_rate	bigint	Receiving rate of the communications library on the node, in KB/s
txkbyte_rate	bigint	Sending rate of the communications library on the node, in KB/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the libcomm process, in bytes
memkbyte_libpq	bigint	Communication memory size of the libpq process, in bytes
used_pm	integer	Real-time usage of the postmaster thread
used_sflow	integer	Real-time usage of the gs_sender_flow_controller thread
used_rflow	integer	Real-time usage of the gs_receiver_flow_controller thread
used_rloop	integer	Highest real-time usage among multiple gs_receivers_loop 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 13-145 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 KB/s
txkbyte_rate	bigint	Sending rate of the communications library on the node, in KB/s
buffer	bigint	Size of the buffer of the Cmailbox

Name	Type	Description
memkbyte_l ibcomm	bigint	Communication memory size of the libcomm process, in bytes
memkbyte_l ibpq	bigint	Communication memory size of the libpq process, in bytes
used_pm	integer	Real-time usage of the postmaster thread
used_sflow	integer	Real-time usage of the gs_sender_flow_controller thread
used_rflow	integer	Real-time usage of the gs_receiver_flow_controller thread
used_rloop	integer	Highest real-time usage among multiple gs_receivers_loop 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 13-146 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: • Catch-up state. • Consistent streaming state.

Name	Type	Description
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 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: • Asynchronous replication. • Synchronous replication. • Potential synchronization.

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 13-147 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: • Catch-up state. • Consistent streaming state.

Name	Type	Description
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 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: • Asynchronous replication. • Synchronous replication. • Potential synchronization.

13.2.12.3 REPLICATION_SLOTS

REPLICATION_SLOTS displays replication slot information.

Table 13-148 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. • physical: physical replication slot. • logical: logical replication slot.
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. • t (true): yes • f (false): no

Name	Type	Description
xmin	xid	The earliest transaction that the database must reserve for the replication slot. VACUUM cannot remove a tuple that is deleted by its subsequent transactions.
catalog_xmin	xid	The earliest transaction that affects the system catalog and must be reserved by the database for the replication slot. VACUUM cannot remove a system catalog tuple that is deleted by its subsequent transactions.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	boolean	Lab features.

13.2.12.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS displays information about replicated slots on each node in the cluster.

Table 13-149 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. • physical: physical replication slot. • logical: logical replication slot.
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. • t (true): yes • f (false): no

Name	Type	Description
x_min	xid	The earliest transaction that the database must reserve for the replication slot. VACUUM cannot remove a tuple that is deleted by its subsequent transactions.
catalog_xmin	xid	The earliest transaction that affects the system catalog and must be reserved by the database for the replication slot. VACUUM cannot remove a system catalog tuple that is deleted by its subsequent transactions.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	boolean	Lab features.

13.2.12.5 BGWRITER_STAT

BGWRITER_STAT displays statistics about the background writer process's activities.

Table 13-150 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 spent on performing write operations during checkpointing (unit: ms).
checkpoint_sync_time	double precision	Total time spent on performing sync operations during checkpointing (unit: ms).
buffers_checkpoint	bigint	Number of buffers written during checkpointing.
buffers_clean	bigint	Number of buffers written by the BGWRITER thread.
maxwritten_clean	bigint	Number of times that the BGWRITER thread stops cleanup scanning because the number of pages refreshed in a single scan is too large.
buffers_backend	bigint	Number of buffers written directly by the backend.

Name	Type	Description
buffers_back end_fsync	bigint	Number of times that the backend thread automatically executes fsync. Generally, fsync is called by the BGWRITER thread.
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time when the statistics of the current row were reset last time.

13.2.12.6 GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT displays statistics about the bgwriter thread's activities on all instances.

Table 13-151 GLOBAL_BGWRITER_STAT columns

Name	Type	Description
node_name	name	Instance 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 spent on performing write operations during checkpointing (unit: ms).
checkpoint_sync_time	double precision	Total time spent on performing sync operations during checkpointing (unit: ms).
buffers_checkpoint	bigint	Number of buffers written during checkpointing.
buffers_clean	bigint	Number of buffers written by the BGWRITER thread.
maxwritten_clean	bigint	Number of times that the BGWRITER thread stops cleanup scanning because the number of pages refreshed in a single scan is too large.
buffers_backend	bigint	Number of buffers written directly by the backend.

Name	Type	Description
buffers_backend_fsync	bigint	Number of times that the backend thread automatically executes fsync. Generally, fsync is called by the BGWRITER thread.
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time when the statistics of the current row were reset last time.

13.2.12.7 POOLER_STATUS

POOLER_STATUS is used to query the cache connection status of the pooler module on the local CN.

Table 13-152 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. • t (true): The connection is in use. • f (false): The connection is not in use.
node_port	integer	Port number of the node connected
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

Name	Type	Description
idx	bigint	Logical connection ID of the connected instance node
streamid	bigint	Stream ID corresponding to each logical connection

13.2.12.8 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 13-153 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	Detects the network status between the current instance and the peer instance: • t (true): The network between instances is normal. • f (false): The network between instances is abnormal.
no_error_occur	Boolean	Pooler connection result between the current instance and the peer instance. • t (true) indicates that the pooler connection is normal. • f (false) indicates that the pooler connection is abnormal.

13.2.12.9 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS displays the information about checkpoints and flushing pages of all instances in the entire cluster.

Table 13-154 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.

13.2.12.10 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS displays the information about doublewrite files of all instances in the entire cluster.

Table 13-155 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
low_threshold_writes	bigint	Number of I/Os for inefficient doublewrite files.

Name	Type	Description
high_threshold_writes	bigint	Number of I/Os for efficient doublewrite files.
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

13.2.12.11 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS displays the information about checkpoints and flushing pages of all instances in the entire cluster.

Table 13-156 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	recovery_lsn of the first dirty page in the dirty page queue of the current instance
queue_rec_lsn	text	recovery_lsn of the dirty page queue of the current instance
current_xlog_insert_lsn	text	The write position of Xlogs in the current instance
ckpt_redo_point	text	Checkpoint of the current instance

13.2.12.12 GLOBAL_POOLER_STATUS

GLOBAL_POOLER_STATUS is used to query the cache connection status of the pooler modules on all CNs.

Table 13-157 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. • t (true): The connection is in use. • f (false): The connection is not in use.
fdsock	bigint	Port file descriptor
remote_pid	bigint	Thread ID of the remote node connected
session_params	text	Session parameter

13.2.12.13 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 13-158 GLOBAL_RECORD_RESET_TIME columns

Name	Type	Description
node_name	text	Node name.
reset_time	timestamp with time zone	Time to be reset.

13.2.12.14 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS displays the replaying of logs about instances in the entire cluster.

Table 13-159 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 log replay in the current instance.
read_data_io_total_dur	bigint	Total I/O duration reads during log replay in the current instance.
write_data_io_counter	bigint	Number of data page I/O writes during log replay in the current instance.
write_data_io_total_dur	bigint	Total I/O duration writes during log replay in the current instance.
process_pending_counter	bigint	Number of synchronization times of log distribution threads during log replay in the current instance.
process_pending_total_dur	bigint	Synchronization duration of log distribution threads during log replay in the current instance.
apply_counter	bigint	Number of synchronization times of replay threads during log replay in the current instance.

Name	Type	Description
apply_total_dur	bigint	Synchronization duration of replay threads during log replay in the current instance.
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 cm_ctl query -rv command to obtain a more accurate replay speed of the standby node.
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 NULL .

13.2.12.15 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS displays log flow control information about the primary and standby nodes.

Table 13-160 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).
target_rto	bigint	Expected flow control time of the standby node specified by the corresponding GUC parameter (unit: s).

Name	Type	Description
current_sleep_time	bigint	Sleep time (in microseconds) required by the host to achieve the expected RTO.

13.2.12.16 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 13-161 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: • r: ordinary table. • t: TOAST table. • i: index.

13.2.12.17 USER_LOGIN

USER_LOGIN records the number of user logins and logouts.

Table 13-162 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 oid in pg_authid .)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

13.2.12.18 SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN records information about user logins and logouts on all CNs.

Table 13-163 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 oid in pg_authid .)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

13.2.12.19 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 13-164 GLOBAL_GET_BGWRITER_STATUS columns

Name	Type	Description
node_name	text	Instance 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_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction.

13.2.12.20 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 13-165 GLOBAL_SINGLE_FLUSH_DW_STATUS columns

Name	Type	Description
node_name	text	Instance 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.

13.2.12.21 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS displays the number of candidate buffers and buffer eviction information of all instances in the database.

Table 13-166 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 13-167 LOCKS columns

Name	Type	Description
locktype	text	Type of the locked object: relation , extend , page , tuple , transactionid , virtualxid , object , userlock , or advisory
database	oid	OID of the database in which the locked object exists. • The OID is 0 if the object is a shared object. • The OID is NULL if the object is a transaction ID.
relation	oid	OID of the relationship targeted by the lock. The value is NULL 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 NULL 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 NULL 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 NULL if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is NULL 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 NULL if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is NULL if the object is not a general database object.
objsubid	smallint	Column number for a column in the table. The value is 0 if the object is some other object type. The value is NULL 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 NULL if the lock is held by a prepared transaction.
sessionid	bigint	ID of the session holding or awaiting this lock The value is NULL if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread
granted	boolean	• The value is TRUE if the lock is a held lock. • The value is FALSE if the lock is an awaited lock.
fastpath	boolean	The value is TRUE if the lock is obtained through fast-path , and is FALSE 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 locktag_decode() 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 13-168 GLOBAL_LOCKS columns

Name	Type	Description
node_name	name	Node name

Name	Type	Description
locktype	text	Type of the locked object: relation , extend , page , tuple , transactionid , virtualxid , object , userlock , or advisory
database	oid	OID of the database in which the locked object exists. - The OID is 0 if the object is a shared object. - The OID is NULL if the object is a transaction ID.
relation	oid	OID of the relationship targeted by the lock. The value is NULL 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 NULL 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 NULL 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 NULL if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is NULL if the object is not a transaction ID.
classid	oid	OID of the system catalog that contains the object. The value is NULL if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is NULL if the object is not a general database object.
objsubid	smallint	Column number for a column in the table. The value is 0 if the object is some other object type. The value is NULL 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 NULL 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 NULL 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	- The value is TRUE if the lock is a held lock. - The value is FALSE if the lock is an awaited lock.
fastpath	boolean	The value is TRUE if the lock is obtained through fast-path , and is FALSE 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 locktag_decode() function.

13.2.14 Wait Event

13.2.14.1 WAIT_EVENTS

WAIT_EVENTS displays statistics about wait events on the current node.
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 13-169 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: μ s)
avg_wait_time	bigint	Average waiting time (unit: μ s)
max_wait_time	bigint	Maximum waiting time (unit: μ s)
min_wait_time	bigint	Minimum waiting time (unit: μ 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. To query a view, you must have the **sysadmin** or **monitor admin** permission.

Table 13-170 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: μ s)
avg_wait_time	bigint	Average waiting time (unit: μ s)
max_wait_time	bigint	Maximum waiting time (unit: μ s)
min_wait_time	bigint	Minimum waiting time (unit: μ s)
last_updated	timestamp with time zone	Last time when the event was updated

13.2.15 Configuration

13.2.15.1 CONFIG_SETTINGS

CONFIG_SETTINGS displays information about parameters of the running database.

Table 13-171 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 internal , postmaster , sighup , backend , superuser , and user
vartype	text	Parameter type, including bool , enum , integer , real , or string
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 null .
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is null .
enumvals	text[]	Valid values of an enum parameter. If the parameter type is not enum, the value of this column is null .
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 null .
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 null .

13.2.15.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS displays information about parameters of running databases on each node.

Table 13-172 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 internal , postmaster , sighup , backend , superuser , and user
vartype	text	Parameter type, including bool , enum , integer , real , or string
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 null .
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is null .
enumvals	text[]	Valid values of an enum parameter. If the parameter type is not enum, the value of this column is null .
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 null .
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 null .

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. If **enable_resource_record** is set to **on**, the system imports records from GS_WLM_OPERATOR_HISTORY to this system catalog every 3 minutes. You are advised not to enable this function, because it occupies storage space and affects performance.

Table 13-173 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.

Name	Type	Description
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 DN.
warning	text	Warning. The following warnings are displayed: • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

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.

If **enable_resource_record** is set to **on**, the records in the view are dumped to the GS_WLM_OPERATOR_INFO system catalog every 3 minutes and deleted from the view. If **enable_resource_record** is set to **off**, the retention period of records in the view is 3 minutes. The recorded data is the same as that of GS_WLM_OPERATOR_INFO.

13.2.16.3 OPERATOR_RUNTIME

OPERATOR_RUNTIME displays information about operators of the jobs that are being executed by the current user.

Table 13-174 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

Name	Type	Description
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 finished or running .
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's
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)
max_cpu_time	bigint	Maximum execution time of the operator on all DN's (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on all DN's (unit: ms)
cpu_skew_percent	integer	Skew of the execution time among DN's

Name	Type	Description
warning	text	Warning. The following warnings are displayed: • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

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 13-175 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 DN's (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among each DN's
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)
max_cpu_time	bigint	Maximum execution time of the operator on all DN's (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on all DN's (unit: ms)
cpu_skew_percent	integer	Skew of the execution time among DN's
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 CN's.

Data is dumped from the kernel to the system catalog
GS_WLM_OPERATOR_INFO.

If the GUC parameter **enable_resource_record** is set to **on**, the records in **GS_WLM_OPERATOR_HISTORY** are imported to the system catalog **GS_WLM_OPERATOR_INFO** every 3 minutes. This view is an aggregated view for

querying the system catalog `GS_WLM_OPERATOR_INFO` from all CNs. Columns in this view are the same as those in [Table 13-175](#).

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 13-176 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 finished or running .
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 DN's
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)

Name	Type	Description
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: • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

13.2.17 Global Plan Cache

13.2.17.1 LOCAL_PLANCACHE_STATUS

LOCAL_PLANCACHE_STATUS displays the status of the GPC plan cache on the current node.

Table 13-177 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

Name	Type	Description
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 0 .

13.2.17.2 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](#).

13.2.18 RTO & RPO

13.2.18.1 global_rto_status

Displays log flow control information about the primary and standby nodes (except the current node and DNs).

Table 13-178 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: μ s) required to reach the expectation

13.2.18.2 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 13-179 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).
rto_sleep_time	int	RTO flow control information, that is, expected sleep time (unit: μ 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: μ s) required by xlogInsert on the host to reach the specified RPO.

13.2.19 Deprecated Views

13.2.19.1 Query

The following views have been deprecated in the latest version of GaussDB:

- GS_SLOW_QUERY_INFO
- GS_SLOW_QUERY_HISTORY
- GLOBAL_SLOW_QUERY_HISTORY
- GLOBAL_SLOW_QUERY_INFO

13.2.19.2 Global Plancache

The following views have been deprecated in the latest version of GaussDB:

- LOCAL_PREPARE_STATEMENT_STATUS
- GLOBAL_PREPARE_STATEMENT_STATUS

14 Logical Replication

14.1 Logical Decoding

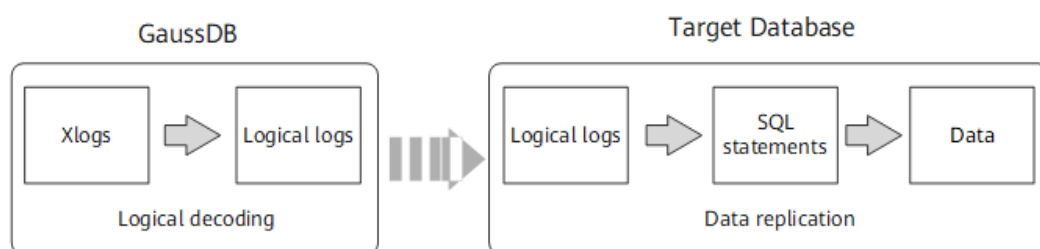
14.1.1 Overview

In GaussDB:

- Data is periodically synchronized to heterogeneous databases (such as Oracle Database) using a data migration tool. Real-time data replication is not supported. Therefore, real-time data synchronization to heterogeneous databases is not satisfied.
- The standby cluster requires that the numbers of CNs and DN and the instance deployment mode be consistent with those in the primary cluster. When the standby cluster is restored, read and write operations cannot be performed, and replication latency is relatively high.

Based on the above two points, GaussDB provides the logical decoding function to generate logical logs by decoding Xlogs. A target database parses logical logs to replicate data in real time. For details, see [Figure 14-1](#). Logical replication reduces the restrictions on target databases, allowing for data synchronization between heterogeneous databases and homogeneous databases with different forms. It allows data to be read and written during data synchronization on a target database, reducing the data synchronization latency.

Figure 14-1 Logical replication



Logical replication consists of logical decoding and data replication. Logical decoding outputs logical logs by transaction. The database service or middleware

parses the logical logs to implement data replication. Currently, GaussDB supports only logical decoding. Therefore, this section involves only logical decoding.

Function

Logical decoding provides basic transaction decoding capabilities for logical replication. GaussDB uses SQL functions for logical decoding. This method features easy function calling, requires no tools to obtain logical logs, and provides specific APIs for interconnecting with external replay tools, saving the need of additional adaptation.

Logical logs are output only after transactions are committed because they use transactions as the unit and logical decoding is driven by users. Therefore, to prevent Xlogs from being recycled by the system when transactions start and prevent required transaction information from being recycled by VACUUM, GaussDB introduces logical replication slots to block Xlog recycling.

A logical replication slot means a stream of changes that can be re-executed in other clusters in the order they were generated in the original cluster. Each logical replication slot is maintained by the person who obtains the corresponding logical logs.

Prerequisites

- Currently, logical logs are extracted from DNs. If logical replication is performed, SSL connections must be used. Therefore, ensure that the **ssl** parameter on the corresponding DN is set to **on**.

NOTE

For security purposes, ensure that SSL connections are enabled.

- The GUC parameter **wal_level** is set to **logical**.
- The GUC parameter **max_replication_slots** is set to a value greater than or equal to the sum of physical replication slots, backup slots, and logical replication slots required by each DN.

NOTE

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.
- A user needs to connect to a database through a DN port before using SQL functions to perform logical decoding. If a CN port is used to connect to the database, **EXECUTE DIRECT ON (datanode_name) 'statement'** is needed to execute SQL functions.
- Only the initial user and users with the REPLICATION permission can perform this operation. When separation of duties is disabled, database administrators can perform logical replication operations. When separation of duties is enabled, database administrators are not allowed to perform logical replication operations.
- Sufficient memory resources must be configured for parallel decoding. The memory usage consists of the following parts:

- Memory for concatenating TOAST tuples and hash tables = Number of concurrent transactions x Average tuple size. The current version is not controlled by the flushing logic. A large amount of memory may be occupied when TOAST tuples are processed.
- Queue memory = Queue length x Decoding parallelism degree x Average tuple size x 2 (including the read log queue and decoding result queue). The queue memory is controlled by the queue length and decoding parallelism degree.
- Memory for sending and re-ordering parallel decoding logs = Number of concurrent transactions x Number of tuples modified by transactions x Average tuple size. The upper limit can be controlled by the decoding configuration options **max-txn-in-memory** and **max-reorderbuffer-in-memory**.
- Memory for storing transaction metadata = Number of concurrent transactions x Metadata size of each transaction (about 300 bytes).

Precautions

- Logical decoding does not support DDL statements.
- Decoded data may be lost when a specific DDL statement (for example, to truncate an ordinary table or exchange data between partitioned tables) is executed.
- Decoding of DML operations on data page replication is not supported.
- Distributed transactions cannot be decoded. Currently, decoding is performed by DN, which cannot ensure consistency in distributed transaction decoding.
- After a DDL statement (for example, ALTER TABLE) is executed, the physical logs that are not decoded before the DDL statement execution may be lost.
- Online cluster scale-out is not allowed during logical decoding.
- The size of a single tuple cannot exceed 1 GB, and decoded data may be larger than inserted data. Therefore, it is recommended that the size of a single tuple be less than or equal to 500 MB.
- GaussDB supports the following data types for decoding: INTEGER, BIGINT, SMALLINT, TINYINT, SERIAL, SMALLSERIAL, BIGSERIAL, FLOAT, DOUBLE PRECISION, DATE, TIME[WITHOUT TIME ZONE], TIMESTAMP[WITHOUT TIME ZONE], CHAR(n), VARCHAR(n), and TEXT.
- If the SSL connection is required, ensure that the GUC parameter **ssl** is set to **on**.
- The name of a logical replication slot can contain no more than 64 characters, including lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). In addition, periods (. or ..) cannot be used as the name of a logical replication slot independently.
- After the database where a logical replication slot resides is deleted, the replication slot becomes unavailable and needs to be manually deleted.
- Interval partitioned tables cannot be replicated.
- To decode multiple databases, you need to create a streaming replication slot in each database and start decoding. Logs need to be scanned for decoding of each database.
- Forcible switchover is not supported. After forcible switchover, you need to export all data again.

- After a DDL statement is executed in a transaction, the DDL statement and subsequent statements are not decoded.
- To perform decoding on the standby node, set the GUC parameter **enable_slot_log** to **on** on the corresponding host.
- During decoding on the standby node, the decoded data may increase during switchover and failover, which needs to be manually filtered out. When the quorum protocol is used, switchover and failover should be performed on the standby node that is to be promoted to primary, and logs must be synchronized from the primary node to the standby node.
- The same replication slot for decoding cannot be used between the primary node and standby node or between different standby nodes at the same time. Otherwise, data inconsistency occurs.
- Replication slots can only be created or deleted on hosts.
- After the database is restarted due to a fault or the logical replication process is restarted, duplicate decoded data may exist. You need to filter out the duplicate data.
- If the computer kernel is faulty, garbled characters may be displayed during decoding, which need to be manually or automatically filtered out.
- Currently, the logical decoding on the standby node does not support enabling the ultimate RTO.
- Ensure that the long transaction is not started during the creation of the logical replication slot. If the long transaction is started, the creation of the logical replication slot will be blocked.
- To parse the UPDATE and DELETE statements of an Astore table, you need to configure the **REPLICA IDENTITY** attribute for the table. If the table does not have a primary key, set the **REPLICA IDENTITY** attribute to **FULL**. For details, see [REPLICA IDENTITY { DEFA...](#)
- Do not perform operations on the replication slot on other nodes when the logical replication slot is in use. To delete a replication slot, stop decoding in the replication slot first.
- Considering that the target database may require the system status information of the source database, logical decoding automatically filters only logical logs of system catalogs whose OIDs are less than 16384 in pg_catalog and pg_toast schemas. If the target database does not need to copy the content of other related system catalogs, the related system catalogs need to be filtered during logical log replay.
- 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**.
- After a logical replication slot is used up, delete it in time. Otherwise, Xlog recycling will be blocked.
- If a transaction has too many sub-transactions, too many files are flushed to disks. To exit decoding, you need to run the SQL function pg_terminate_backend (walsender thread ID for logical decoding) to manually stop decoding. In addition, the exit delay increases by about 1 minute per 300,000 sub-transactions. Therefore, when logical decoding is enabled, if the

number of sub-transactions of a transaction reaches 50,000, a WARNING log is generated.

- When a transaction generates a large number of sub-transactions that need to be flushed to disks, the number of opened file handles may exceed the upper limit. In this case, set **max_files_per_process** to a value greater than twice the upper limit of sub-transactions.

SQL Function Decoding Performance

In the Benchmarksql-5.0 with 100 warehouses, when `pg_logical_slot_get_changes` is used:

- If 4000 lines of data (about 5 MB to 10 MB logs) are decoded at a time, the decoding performance ranges from 0.3 MB/s to 0.5 MB/s.
- If 32000 lines of data (about 40 MB to 80 MB logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
- If 256000 lines of data (about 320 MB to 640 MB logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
- If the amount of data to be decoded at a time still increases, the decoding performance is not significantly improved.

If `pg_logical_slot_peek_changes` and `pg_replication_slot_advance` are used, the decoding performance is 30% to 50% lower than that when `pg_logical_slot_get_changes` is used.

14.1.2 Logical Decoding Options

General Options (Both serial decoding and parallel decoding can be configured, but the settings may be invalid. For details, see the description of related options.)

- **include-xids:**
Specifies whether the decoded **data** column contains XID information.
Valid value: **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.
Valid value: **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.
Valid value: **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.

Valid value: **0** and **1**. The default value is **1**.

- **0**: Non-local logs and local logs are decoded.
- **1**: Only local logs are decoded.

- **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. For example:

```
select * from pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list',  
'public.t1,public.t2,*,t3,my_schema.*');
```

- **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 disks.

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 disks.

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. 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.

Valid value: **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: a string, which specifies the OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the OIDs exist.

- **exclude-users:**

Name list of blacklisted users.

Value range: a string, which specifies the names of blacklisted users. Multiple names are separated by commas (,). The system does not check whether the names exist.

- **dynamic-resolution:**

Specifies whether to dynamically parse the names of blacklisted users.

Valid value: **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**.

- **standby-connection:**

Specifies whether to restrict decoding only on the standby node. This option is valid only for streaming decoding.

Value range: Boolean. The default value is **false**.

- **true**: 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.
- **false**: The primary or standby node can be connected for decoding.

- **sender-timeout:**

Specifies the heartbeat timeout threshold between the kernel and the client. This option is valid only for streaming decoding. If no message is received from the client within the period, the logical decoding stops and disconnects from the client. The unit is ms.

Value range: an integer ranging from 0 to 2147483647. The default value depends on the value of the GUC parameter **logical_sender_timeout**.

- **parallel-decode-num:**

Specifies the number of decoder threads for parallel decoding. This option is valid only for streaming decoding. When the system function is called, this option is invalid and only the value range is verified.

Value range: 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**.

NOTICE

If **parallel-decode-num** is not set (the default value is **1**) or is explicitly set to **1**, the options in the following "Parallel decoding" cannot be configured.

Serial Decoding

- **force-binary:**

Specifies whether to output the decoding result in binary format and display different behaviors in different scenarios.

- For system functions `pg_logical_slot_get_binary_changes` and `pg_logical_slot_peek_binary_changes`:

Value range: Boolean. The default value is **false**. The value is meaningless. The decoding result is always output in binary format.

- For system functions `pg_logical_slot_get_changes`, `pg_logical_slot_peek_changes`, and `pg_logical_get_area_changes`:

Value range: Boolean. The value is fixed to **false**. The decoding result is always output in text format.

- For streaming decoding:

Value range: Boolean. The default value is **false**. The value is meaningless. The decoding result is always output in text format.

Parallel Decoding

- The following configuration options are set only for streaming decoding.

- **decode-style:**

Specifies the decoding format.

Valid value: 'j', 't', or 'b' of the char type, indicating the JSON, TEXT, or binary format, respectively. The default value is 'b', indicating binary decoding.

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 the letter described in 3 is **B**:
 1. The next eight bytes (uint64) indicate the CSN.
 2. The next eight bytes (uint64) indicate **first_lsn**.
 3. (Optional) If the next one-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. Because there may still be a decoding statement subsequently, a one-byte letter **P** or **F** is used as a separator between statements. **P** indicates that there are still decoded statements in this batch, and **F** indicates that this batch is completed.
5. If the letter described in 3 is **C**:
 1. (Optional) If the next one-byte letter is **X**, the following eight bytes (uint64) indicate the XID.
 2. (Optional) If the next one-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 one-byte letter is **P**, the batch still needs to be decoded. If the letter is **F**, the batch decoding ends.
6. If the letter described in 3 is **I**, **U**, or **D**:
 1. The next two bytes (uint16) indicate the length of the schema name.
 2. The schema name is read based on the preceding length.
 3. The next 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 one-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 next 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.
- **sending-batch:**
Specifies whether to send messages in batches.
Valid value: **0** or **1** of the int type. The default value is **0**.
 - **0**: The decoding results are sent one by one.
 - **1**: When the accumulated size of decoding results reaches 1 MB, decoding results are sent in batches.In the scenario where batch sending is enabled, if the decoding format is 'j' or 't', before each original decoding statement, a uint32 number is added indicating the length of the decoding result (excluding the current uint32 number), and a uint64 number is added indicating the LSN corresponding to the current decoding result.
 - **parallel-queue-size:**
Specifies the length of the queue for interaction between parallel logical decoding threads.
Value range: an integer ranging from 2 to 1024. The value must be an integer power of 2. The default value is **128**.
The queue length is positively correlated with the memory usage during decoding.

14.1.3 Logical Decoding by SQL Function Interfaces

In GaussDB, you can call SQL functions to create, delete, and update logical replication slots, as well as obtain decoded transaction logs.

Procedure

Step 1 Log in to any host in the GaussDB cluster as a user with the REPLICATION permission.

Step 2 Run the following command to connect to the database through a DN port:

```
gsql -U user1 -d gaussdb -p 40000 -r
```

In the preceding command, **user1** indicates the username, **gaussdb** indicates the name of the database to be connected, and **40000** indicates the database DN port number. You can replace them as required. Replication slots are created on DNs. Therefore, you need to connect to a database through a DN port.

Step 3 Create a logical replication slot named **slot1**.

```
openGauss=# SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
slotname | xlog_position
-----+-----
slot1    | 0/601C150
(1 row)
```

Step 4 Create a table **t** in the database and insert data into it.

```
openGauss=# CREATE TABLE t(a int PRIMARY KEY, b int);
openGauss=# INSERT INTO t VALUES(3,3);
```

Step 5 Read the decoding result of **slot1**. The number of decoded records is 4096.

```
openGauss=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096);
location | xid | data
-----+-----
0/601C188 | 1010023 | BEGIN 1010023
0/601ED60 | 1010023 | COMMIT 1010023 CSN 1010022
0/601ED60 | 1010024 | BEGIN 1010024
0/601ED60 | 1010024 | {"table_name":"public.t","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
0/601EED8 | 1010024 | COMMIT 1010024 CSN 1010023
(5 rows)
```

Step 6 Delete the logical replication slot **slot1**.

```
openGauss=# SELECT * FROM pg_drop_replication_slot('slot1');
pg_drop_replication_slot
-----
(1 row)
```

----End

14.1.4 Replicating Data Using the Logical Replication Tool

Currently, the SDR and DRS tools support GaussDB logical replication. The replication tools extract logical logs from GaussDB and replay them on the peer database.

For details about the code of the replication tool that uses JDBC to connect to the database, see [Example: Logic Replication Code](#).

15 GTM Mode

To meet different concurrency and consistency requirements, GaussDB provides three GTM modes: GTM, GTM-Lite, and GTM-Free. The main difference between the three modes lies in the pressure of the central transaction management node GTM and the transaction processing process. In the GTM mode, the central transaction processing node has the highest pressure and is prone to become a performance and concurrency bottleneck. In GTM-Lite mode, the pressure on the central transaction processing node is reduced, the transaction processing process is further optimized, and the GTM performance and concurrency bottleneck are reduced. In this way, the transaction processing capability is improved to a great extent while ensuring consistency. In GTM-Free mode, the central transaction management node does not participate in transaction management, eliminating the GTM single-point bottleneck and achieving higher transaction processing performance. However, in terms of consistency, all transactions can be completed to ensure external read consistency. Strong consistency read of distributed transactions is not supported, transaction consistency that depends on query results, such as insert into select * from, is not supported.

The current version does not support the switchover between the three modes. You are advised to use the default GTM mode during installation. The GTM mode can remain unchanged before and after the upgrade.

Related GUC parameters include `enable_gtm_free` and `gtm_option`. You can run the following show statement in `gsql` to query the current GTM mode:

```
SHOW enable_gtm_free;  
SHOW gtm_option;
```

The method of determining the mode is as follows:

- GTM mode: **`enable_gtm_free=off`** and **`gtm_option=0`**
- GTM-lite mode: **`enable_gtm_free=off`** and **`gtm_option=1`**
- GTM-free mode: **`enable_gtm_free=on`** or **`gtm_option=2`**

In GTM-lite and GTM modes, GaussDB supports strong consistency and complete syntaxes for distributed transactions. In GTM-free mode, eventual consistent execution and concurrency control are used for distributed transactions. Therefore, the usage scenarios and methods of some syntaxes are restricted. If the restricted syntaxes are required due to service requirements, refactoring is required based on the understanding of the eventual consistent behavior. The involved restricted syntaxes and refactoring suggestions are as follows:

1. General principle: Specify a proper distribution key (by using DISTRIBUTE BY) for all user tables.
 - You are advised to distribute data evenly.
 - You are advised to use the join condition in a query as the distribution key to ensure that the join query does not cause data flow between DNs.
 - You are advised to select the primary key of the table as the distribution key.
2. SELECT:
 - During table query, the WHERE condition must contain the equivalent query condition of all distribution keys.
 - 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.

3. DML:

By default, cross-node transactions are not supported. If the executed DML statement contains cross-node transactions, an error is reported. There are two scenarios:

- a. If a user statement is split into multiple independent statements in the database, the error message "INSERT/UPDATE/DELETE/MERGE contains multiple remote queries under GTM-free mode Unsupport DML two phase commit under gtm free mode. modify your SQL to generate light-proxy or fast-query-shipping plan" is displayed. In this case, you need to modify the statement to execute it on a single node.

For example:

```
insert into t select * from b where b.c = xx;
```

Assume that the distribution keys of the **t** and **b** tables are different, and the WHERE condition filters out only one data record. If **enable_stream_operator** is disabled, the preceding query is split into two independent statements and executed in serial mode. First, run **select * from b where b.c = xx** to extract data from a DN to the target record. Then, run **insert into t** to deliver the extracted target record to another DN to complete the insertion. In GTM-free mode, the preceding error is reported when such a statement is executed. Similar errors may also be reported for **create table as select * from** and DELETE, JOIN, and INSERT statements with subqueries.

Refactoring solution: Before statement execution, add the **set enable_stream_operator=on** command to enable the streaming operator so that service statements can be pushed down for execution.

- b. If a user statement is executed on multiple nodes in the database, the error message "Your SQL needs more than one datanode to be involved in" will be displayed. In this case, you are advised to modify the statement so that it can be executed on a single node.

For example:

```
insert into t values(3,3),(1,1);
```

If (3,3) and (1,1) are distributed on different DNs, the execution of the preceding statement involves two DNs. In GTM-free mode, the preceding error is reported when such a statement is executed.

Refactoring method: If the preceding statement needs to be executed on multiple nodes, add a hint to the statement to prevent errors. The hint is as follows:

```
insert /*+ multinode */ into t values(3,3),(1,1);
```

There are similar constraints on the DELETE and UPDATE statements. You are advised to add the equivalent filter condition of the distribution key to the WHERE condition in the DELETE and UPDATE statements.

4. It is recommended that **application_type** be set to **perfect_sharding_type** in the JDBC connection string in the development phase. In this way, errors are reported for all SQL statements for cross-node read and write operations, prompting developers to optimize statements as soon as possible.

16 Materialized View

A materialized view is a special physical table, which is relative to an ordinary view. An ordinary view is a virtual table and has many application limitations. Any query on a view is actually converted into a query on an SQL statement, and performance is not actually improved. The materialized view actually stores the results of the statements executed by the SQL statement, and is used to cache the results.

16.1 Complete-refresh Materialized View

16.1.1 Overview

Complete-refresh materialized views refer to the created materialized views that support only complete refresh but do not support fast refresh.

The syntax for creating a complete-refresh materialized view is the same as the CREATE TABLE AS syntax. You cannot specify a NodeGroup to create a complete-refresh materialized view.

16.1.2 Usage

Syntax

- Create a complete-refresh materialized view.
`CREATE MATERIALIZED VIEW [ view_name ] AS { query_block };`
- Completely refresh a materialized view.
`REFRESH MATERIALIZED VIEW [ view_name ];`
- Delete a materialized view.
`DROP MATERIALIZED VIEW [ view_name ];`
- Query a materialized view.
`SELECT * FROM [ view_name ];`

Examples

```
-- Prepare data.  
CREATE TABLE t1(c1 int, c2 int);  
INSERT INTO t1 VALUES(1, 1);  
INSERT INTO t1 VALUES(2, 2);
```

```
-- Create a complete-refresh materialized view.
openGauss=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
count
-----
      2
(1 row)

-- Insert data into the base table in the materialized view again.
openGauss=# INSERT INTO t1 VALUES(3, 3);

-- Completely refresh a complete-refresh materialized view.
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
count
-----
      3
(1 row)

-- Delete a materialized view.
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

16.1.3 Support and Constraints

Supported Scenarios

- Generally, the query scope supported by complete-refresh materialized views is the same as that supported by the CREATE TABLE AS statement.
- The distribution key can be specified when a complete-refresh materialized view is created.
- Indexes can be created in a complete-refresh materialized view.
- ANALYZE and EXPLAIN are supported.

Unsupported Scenarios

- Complete-refresh materialized views do not support node groups.
- Materialized views cannot be added, deleted, or modified. Only query statements are supported.

Constraints

- The base table used to create a complete-refresh materialized view must be defined on all DNs, and the node group to which the base table belongs must be an installation group.
- When a complete-refresh materialized view is refreshed or deleted, a high-level lock is added to the base table. If the definition of a materialized view involves multiple tables, pay attention to the service logic to avoid deadlock.

16.2 Fast-refresh Materialized View

16.2.1 Overview

Fast-refresh materialized views allow for incremental refreshes as the name suggests. You must manually execute statements to refresh the incremental data of materialized views over a certain period.

The difference between the incremental and the complete-refresh materialized views is that the fast-refresh materialized view supports only a small number of scenarios. Currently, only base table scanning statements or UNION ALL can be used to create materialized views.

16.2.2 Usage

Syntax

- Create a fast-refresh materialized view.
`CREATE INCREMENTAL MATERIALIZED VIEW [ view_name ] AS { query_block };`
- Completely refresh a materialized view.
`REFRESH MATERIALIZED VIEW [ view_name ];`
- Fastly refresh a materialized view.
`REFRESH INCREMENTAL MATERIALIZED VIEW [ view_name ];`
- Delete a materialized view.
`DROP MATERIALIZED VIEW [ view_name ];`
- Query a materialized view.
`SELECT * FROM [ view_name ];`

Examples

```
-- Prepare data.
CREATE TABLE t1(c1 int, c2 int);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES(2, 2);

-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM t1;
CREATE MATERIALIZED VIEW

-- Insert data.
openGauss=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

-- Fast refresh a materialized view.
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
c1 | c2
----+----
 1 |  1
 2 |  2
 3 |  3
(3 rows)

-- Insert data.
openGauss=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

-- Completely refresh the materialized view.
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW
```

```
-- Query the materialized view result.
openGauss=# select * from mv;
c1 | c2
----+----
 1 |  1
 2 |  2
 3 |  3
 4 |  4
(4 rows)

-- Drop a materialized view.
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

16.2.3 Support and Constraints

Supported Scenarios

- Supports statements for querying a single table.
- Supports UNION ALL for querying multiple single tables.
- Creates an index in a materialized view.
- Performs the ANALYZE operation in the materialized view.
- Creates a fast-refresh materialized view based on the node group of base tables. (Check whether the base tables are in the same node group and create the fast-refresh materialized view based on the node group).

Unsupported Scenarios

- Materialized views do not support the Stream plan, multi-table join plan, or subquery plan.
- Except for a few ALTER operations, most DDL operations cannot be performed on base tables in materialized views.
- A distribution key of a materialized view cannot be specified when the materialized view is created.
- Materialized views cannot be added, deleted, or modified. Only query statements are supported.
- Materialized views cannot be created using a temporary, hash bucket, unlogged, or partitioned table. Only the hash distributed table is supported.
- Materialized views cannot be created in nested mode (that is, a materialized view cannot be created in another materialized view).
- Only row-store tables are supported.
- Materialized views of the UNLOGGED type are not supported, and the WITH syntax is not supported.

Constraints

- If the materialized view is defined as UNION ALL, each subquery must use a different base table and the distribution key of each base table must be the same. The distribution key of the materialized view is automatically deduced and is the same as that of each base table.
- The columns defined in the materialized view must contain all distribution keys in the base table.

- When a fast-refresh materialized view is created, completely refreshed, or deleted, a high-level lock is added to the base table. If the materialized view is defined as UNION ALL, pay attention to the service logic to avoid deadlock.

17 Error Log Reference

17.1 Kernel Error Information

ERRMSG: "unsupported syntax: ENCRYPTED WITH in this operation"

SQLSTATE: 42601

CAUSE: "client encryption feature is not supported this operation."

ACTION: "Check client encryption feature whether supported this operation."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant options cannot be granted to public."

ACTION: "Grant grant options to roles."

ERRMSG: "unrecognized object kind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unrecognized GrantStmt.targtype: %d"

SQLSTATE: XX004

CAUSE: "The target type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported target types."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant to public operation is forbidden in security mode."

ACTION: "Don't grant to public in security mode."

ERRMSG: "unrecognized object type"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid grant/revoke operation"

SQLSTATE: 0LP01

CAUSE: "Column privileges are only valid for relations in GRANT/REVOKE."

ACTION: "Use the column privileges only for relations."

ERRMSG: "invalid AccessPriv node"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined client master key"

SQLSTATE: 42705

CAUSE: "The client master key does not exist."

ACTION: "Check whether the client master key exists."

ERRMSG: "undefined column encryption key"

SQLSTATE: 42705

CAUSE: "The column encryption key does not exist."

ACTION: "Check whether the column encryption key exists."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "The large object does not exist."

ACTION: "Check whether the large object exists."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'schemas' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'roles' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "option '%s' not recognized"

SQLSTATE: 42601

CAUSE: "The option in ALTER DEFAULT PRIVILEGES statement is not supported."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for ALTER DEFAULT PRIVILEGES."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid alter default privileges operation"

SQLSTATE: 0LP01

CAUSE: "Default privileges cannot be set for columns."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for default privileges."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "could not find tuple for default ACL %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unexpected default ACL type: %d"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for default privilege."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid object id"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined column"

SQLSTATE: 42703

CAUSE: "The column of the relation does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "column number out of range"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for attribute %d of relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Index type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Composite type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE SEQUENCE only support sequence objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid privilege type USAGE for table"

SQLSTATE: 0LP01

CAUSE: "GRANT/REVOKE TABLE do not support USAGE privilege."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for tables."

ERRMSG: "invalid privilege type %s for column"

SQLSTATE: 0LP01

CAUSE: "The privilege type is not supported for column object."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for column object."

ERRMSG: "cache lookup failed for database %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign-data wrapper %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign server %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for function %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for language %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Grant/revoke on untrusted languages if forbidden."

SQLSTATE: 0LP01

CAUSE: "Grant/revoke on untrusted languages if forbidden."

ACTION: "Support grant/revoke on trusted C languages"

ERRMSG: "cache lookup failed for large object %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for namespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Role %s has not privilege to grant/revoke node group %s."

SQLSTATE: 42501

CAUSE: "Role has not privilege to grant/revoke node group."

ACTION: "Must have sysadmin privilege."

ERRMSG: "Can not grant CREATE privilege on node group %u to role %u in node group %u."

SQLSTATE: 42501

CAUSE: "Role has not privilege to grant CREATE privilege node group."

ACTION: "Must have sysadmin privilege."

ERRMSG: "cache lookup failed for tablespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for type %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cannot set privileges of array types"

SQLSTATE: 0LP01

CAUSE: "Cannot set privileges of array types."

ACTION: "Set the privileges of the element type instead."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE DOMAIN only support domain objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "cache lookup failed for client master key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for column encryption key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for directory %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized privilege type '%s'"

SQLSTATE: 42601

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized privilege: %d"

SQLSTATE: XX004

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized AclResult"

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for column '%s' of relation '%s'"

SQLSTATE: 42501

CAUSE: "Insufficient privileges for the column."

ACTION: "Select the system tables to get the acl of the column."

ERRMSG: "role with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized objkind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for privilege check."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "attribute %d of relation with OID %u does not exist"

SQLSTATE: 42703

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "the column has been dropped"

SQLSTATE: 42703

CAUSE: "The column does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "relation with OID %u does not exist"

SQLSTATE: 42P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "invalid group"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "database with OID %u does not exist"

SQLSTATE: 3D000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "directory with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "function with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "client master key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "language with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "schema with OID %u does not exist"

SQLSTATE: 3F001

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "node group with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "tablespace with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign-data wrapper with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign server with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "type with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "column encryption key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator class with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator family with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search dictionary with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search configuration with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "collation with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "conversion with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "synonym with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "package can not create the same name with schema."

SQLSTATE: 22023

CAUSE: "Package name conflict"

ACTION: "Please rename package name"

ERRMSG: "type is not exists %s."

SQLSTATE: 22023

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "This input type is not supported for tdigest_in()"
SQLSTATE: 0A000
CAUSE: "input type is not supported"
ACTION: "Check tdigest_in syntax to obtain the supported privilege types"

ERRMSG: "Failed to apply for memory"
SQLSTATE: 53200
CAUSE: "palloc failed"
ACTION: "Check memory"

ERRMSG: "Failed to get tde info from relation '%s'. "
SQLSTATE: XX005
CAUSE: "System error."
ACTION: "Contact engineer to support."

ERRMSG: "SPI_connect failed: %s"
SQLSTATE: SP001
CAUSE: "System error."
ACTION: "Analyze the error message before the error"

ERRMSG: "permission denied for terminate snapshot thread"
SQLSTATE: 42501
CAUSE: "The user does not have system admin privilege"
ACTION: "Grant system admin to user"

ERRMSG: "terminate snapshot thread failed"
SQLSTATE: OP001
CAUSE: "Execution failed due to: %s"
ACTION: "check if snapshot thread exists"

ERRMSG: "terminate snapshot thread failed"
SQLSTATE: OP001
CAUSE: "restart wdr snapshot thread timeoutor The thread did not respond to the kill signal"

ACTION: "Check the wdr snapshot thread is restarted"

ERRMSG: "set lockwait_timeout failed"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for create WDR Snapshot"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "WDR snapshot request can not be accepted, please retry later"

SQLSTATE: OP001

CAUSE: "wdr snapshot thread does not exist"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "Cannot respond to WDR snapshot request"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "query(%s) can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create sequence failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check if sequence can be created"

ERRMSG: "update snapshot end time stamp filled"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query(%s) execute failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean table of snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "analyze table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert into tables_snap_timestamp start time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert data failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean snapshot id %lu is failed in snapshot table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "clean snapshot failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "can not create snapshot stat table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create WDR snapshot data table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into tables_snap_timestamp start time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create index failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "analyze table, connection failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "snapshot thread SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "Distributed key column can't be transformed"

SQLSTATE: 42P10

CAUSE: "There is a risk of violating uniqueness when transforming distribution columns."

ACTION: "Change transform column."

ERRMSG: "cannot convert %s to %s"

SQLSTATE: 42804

CAUSE: "There is no conversion path in pg_cast."

ACTION: "Rewrite or cast the expression."

ERRMSG: "create matview on TDE table failed"

SQLSTATE: 0A000

CAUSE: "create materialized views is not supported on TDE table"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "schema name can not same as package"

SQLSTATE: 22023

CAUSE: "schema name conflict"

ACTION: "rename schema name"

ERRMSG: "Unrecognized commandType when checking read-only attribute."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Fail to generate subquery plan."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing qual condition."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing const parameters."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "SELECT FOR UPDATE/SHARE is not allowed with UNION/INTERSECT/
EXCEPT"

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "GROUP BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "GROUP BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Unrecognized node type when extracting index."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Ordering operator cannot be identified."

SQLSTATE: 42883

CAUSE: "Grouping set columns must be able to sort their inputs."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "DISTINCT cannot be implemented."

SQLSTATE: 0A000

CAUSE: "DISTINCT uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to locate grouping columns."

SQLSTATE: 55000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Resjunk output columns are not implemented."

SQLSTATE: 20000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "PARTITION BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "PARTITION BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "ORDER BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "ORDER BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to deconstruct sort operators into partitioning/ordering operators."

SQLSTATE: D0011

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Pool size should not be zero"

SQLSTATE: 22012

CAUSE: "Compute pool configuration file contains error."

ACTION: "Please check the value of 'pl' in cp_client.conf."

ERRMSG: "Failed to get the runtime info from the compute pool."

SQLSTATE: 22004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Version is not compatible between local cluster and the compute pool."

SQLSTATE: XX008

CAUSE: "Compute pool is not installed appropriately."

ACTION: "Configure compute pool according to manual."

ERRMSG: "No optional index path is found."

SQLSTATE: 01000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "MERGE INTO on replicated table does not yet support using distributed tables."

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Fail to find ForeignScan node!"

SQLSTATE: P0002

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "sql advisor don't support none table, temp table, system table."

SQLSTATE: 42601

CAUSE: "sql advisor don't support none table, temp table, system table."

ACTION: "check query component"

ERRMSG: "Invalid autonomous transaction return datatypes"

SQLSTATE: P0000

CAUSE: "PL/SQL uses unsupported feature."

ACTION: "Contact Huawei Engineer."

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "check table definition"

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "set client_min_messages = info for more details"

ERRMSG: "get gauss home path is NULL"

SQLSTATE: XX005

CAUSE: "gauss home path not set"

ACTION: "check if \$GAUSSHOM is exist"

ERRMSG: "unable to open kms_iam_info.json file"

SQLSTATE: 58P03

CAUSE: "file not exist or broken"

ACTION: "check the kms_iam_info.json file"

ERRMSG: "can not get password plaintext"

SQLSTATE: XX005

CAUSE: "file not exist or broken"

ACTION: "check the password cipher rand file"

ERRMSG: "IAM info json key is NULL"

SQLSTATE: XX005

CAUSE: "IAM info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "get internal password is NULL"

SQLSTATE: XX005

CAUSE: "cipher rand file missing"

ACTION: "check password cipher rand file"

ERRMSG: "KMS info json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "unable to get json file"

SQLSTATE: 58P03

CAUSE: "parse json file failed"

ACTION: "check the kms_iam_info.json file format"

ERRMSG: "get JSON tree is NULL"

SQLSTATE: XX005

CAUSE: "get KMS JSON tree failed"

ACTION: "check input parameter or config.ini file"

ERRMSG: "failed to get json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input parameter or config.ini file"

ERRMSG: "failed to set the value of json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input parameter or config.ini file"

ERRMSG: "http request failed"

SQLSTATE: XX005

CAUSE: "http request error"

ACTION: "check KMS or IAM connect or config parameter"

ERRMSG: "get iam token or iam agency token is NULL"

SQLSTATE: XX005

CAUSE: "connect IAM failed"

ACTION: "check if your env can connect with IAM server"

ERRMSG: "KMS dek json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS return value error"

ACTION: "check KMS config parameter"

ERRMSG: "get kms dek is NULL"

SQLSTATE: XX005

CAUSE: "connect KMS failed"

ACTION: "check if your env can connect with KMS server"

ERRMSG: "get http header is NULL"

SQLSTATE: XX005

CAUSE: "http request failed"

ACTION: "check IAM config parameter"

ERRMSG: "create KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"
ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS dek failed"
SQLSTATE: XX005
CAUSE: "KMS error"
ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS DEK is NULL"
SQLSTATE: XX005
CAUSE: "get KMS dek_plaintext failed"
ACTION: "check KMS network or cipher is right"

ERRMSG: "create matview with TDE failed"
SQLSTATE: 0A000
CAUSE: "TDE feature is not supported for Create materialized views"
ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "failed to add item to the index page"
SQLSTATE: XX002
CAUSE: "System error."
ACTION: "Check WARNINGS for the details."

ERRMSG: "index row size %lu exceeds maximum %lu for index '%s'"
SQLSTATE: 54000
CAUSE: "Values larger than 1/3 of a buffer page cannot be indexed."
ACTION: "Consider a function index of an MD5 hash of the value, or use full text indexing."

ERRMSG: "fail to insert a tuple to an ordered index, the ordered tuple list is corrupted"
SQLSTATE: XX002
CAUSE: "System error."
ACTION: "Contact engineer to support."

ERRMSG: "Tag field is too long."

SQLSTATE: 54000
CAUSE: "Tag buffer overflow."
ACTION: "Shorten tag Key."

17.2 CM Error Information

ERRMSG: "Fail to access the cluster static config file."
SQLSTATE: c3000
CAUSE: "The cluster static config file is not generated or is manually deleted."
ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to open the cluster static file."
SQLSTATE: c3000
CAUSE: "The cluster static config file is not generated or is manually deleted."
ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the cluster static file."
SQLSTATE: c3001
CAUSE: "The cluster static file permission is insufficient."
ACTION: "Please check the cluster static config file."

ERRMSG: "Failed to read the static config file."
SQLSTATE: c1000
CAUSE: "out of memeory."
ACTION: "Please check the system memory and try again."

ERRMSG: "Could not find the current node in the cluster by the node id %u."
SQLSTATE: c3002
CAUSE: "The static config file probably contained content error."
ACTION: "Please check static config file."

ERRMSG: "Failed to open the logic config file."
SQLSTATE: c3000
CAUSE: "The logic config file is not generated or is manually deleted."
ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the logic static config file."

SQLSTATE: c3001

CAUSE: "The logic static config file permission is insufficient."

ACTION: "Please check the logic static config file."

ERRMSG: "Failed to open or read the static config file."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "Log file not found."

ACTION: "Please check the log file."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "The log file permission is insufficient."

ACTION: "please check the log file."

ERRMSG: "Failed to open the dynamic config file '%s'."

SQLSTATE: c3000

CAUSE: "The dynamic config file permission is insufficient."

ACTION: "Please check the dynamic config file."

ERRMSG: "Failed to malloc memory, size = %lu."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "unrecognized AZ name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "unrecognized minorityAz name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "Get GAUSSHOM failed."

SQLSTATE: c3000

CAUSE: "The environment variable('GAUSSHOM') is incorrectly configured."

ACTION: "Please check the environment variable('GAUSSHOM')."

ERRMSG: "Get current user name failed."

SQLSTATE: c3000

CAUSE: "N/A"

ACTION: "Please check the environment."

ERRMSG: "-B option must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-T option must be specified.\n"

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m normal."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one availability zone with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode need not be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -R are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -D are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no operation specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Please check the usage of switchover."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -z cannot be specified at the same time."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-m cannot be specified at the same time with -n or -z."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%d) is invalid."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node is needed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "%s: -C is needed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-z value must be 'ALL' when query mppdb cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-v is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-C is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-Cv is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-L value must be 'ALL' when query logic cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized LC name '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "There is no '%s' information in cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is too long.\n"

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%s) is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R only support when the cluster is single-inst."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-t time is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-votenum is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "too many command-line arguments (first is '%s')."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized operation mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Failed to malloc memory."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open etcd: %s."

SQLSTATE: c4000

CAUSE: "Etcd is abnormal."

ACTION: "Please check the Cluster Status and try again."

ERRMSG: "[PATCH-ERROR] hotpatch command or path set error."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no standby datanode in single node cluster."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed"

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "The option parameter is not specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

18 Configuring GUC Parameters

18.1 Viewing Parameter Values

GaussDB uses a set of default GUC parameters after it is installed. You can modify the parameters to better fit your service scenarios and data volume.

Procedure

Step 1 Connect to the database. For details, see [Using gsql to Connect to an Instance](#).

```
gsql -d postgres -p 8000
```

postgres is the name of the database, and **8000** is the port number of the CN.

If information similar to the following is displayed, the connection succeeds:

```
gsql((GaussDB Kernel V.xxxR.xxxC.xx build f521c606) compiled at 2021-09-16 14:55:22 last mr 6385 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

openGauss=#
```

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:

```
openGauss=# SHOW server_version;
```

server_version indicates the database version.
 - Run the following command to view values of all parameters:

```
openGauss=# SHOW ALL;
```
- Method 2: Query the **pg_settings** view.
 - Run the following command to view the value of a certain parameter:

```
openGauss=# SELECT * FROM pg_settings WHERE NAME='server_version';
```
 - Run the following command to view values of all parameters:

```
openGauss=# SELECT * FROM pg_settings;
```

----End

Example

Check the character encoding type of the client.


```
openGauss=# SHOW client_encoding;
client_encoding
-----
UTF8
(1 row)
```

18.2 Resetting 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 DN nodes at a time, but cannot do the same to other parameters.

For details about parameters in the hosts configuration template, see [GUC Parameters](#).

Setting GUC Parameters

GaussDB provides six types of GUC parameters. For details about parameter types and their setting methods, see [Table 18-1](#).

Table 18-1 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 SHOW command or in the pg_settings view.	None

Parameter Type	Description	Setting Method
POSTMASTER	Database server parameter. It can be set when the database is started or in the configuration file.	Method 1 in Table 18-2 .
SIGHUP	Global database parameter. It can be set when the database is started or be modified later.	Method 1 or 2 in Table 18-2 .
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 Table 18-2 . NOTE 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 Table 18-2 .
USERSET	Common user parameter. It can be set by any user at any time.	Method 1, 2, or 3 in Table 18-2 . NOTE When you set parameters of the USERSET type, the parameter value set using ALTER DATABASE takes precedence over that set using gs_guc . To make the parameter settings of gs_guc take effect, run the alter database xxx reset xxx command to reset the parameters.

You can set GUC parameters in GaussDB using the methods listed in [Table 18-2](#).

Table 18-2 Methods for setting GUC parameters

No.	Setting Method
Method 1	1. Log in to the console. 2. On the Instances page, click the instance name to go to the Basic Information page. 3. In the navigation pane on the left, click Parameters. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the risks in advance and contact customer service for modification. 4. Restart the database to make the setting take effect. NOTE Rebooting instances will interrupt user operations. Plan a proper execution window before the reboot.
Method 2	1. Log in to the console. 2. On the Instances page, click the instance name to go to the Basic Information page. 3. In the navigation pane on the left, click Parameters. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the risks in advance and contact customer service for modification.
Method 3	Modify a session-level parameter. • Set a session-level parameter. <code>openGauss=# SET <i>paraname</i> TO <i>value</i>;</code> Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.

 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.
- When you use method 3 to set a parameter, if the parameter value is an integer, leading zeros will be filtered out. For example, **SET *paraname* TO 008192** and **SET *paraname* TO 8192** have the same effect.

18.3 GUC Parameters

18.3.1 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.

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.

18.3.2 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

NOTICE

This parameter takes effect only after the database restarts.

Value range: a string

Default value: empty

18.3.3 Connection and Authentication

18.3.3.1 Connection Settings

This section describes parameters related to client-server connection modes.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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).

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, *dataPortBase* + 1 needs to be reserved as the port used by internal tools, and *dataPortBase* + 6 needs to be reserved as the communication port of the streaming 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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: $262143 - \text{job_queue_processes} - \text{autovacuum_max_workers} - \text{max_inner_tool_connections} - \text{AUXILIARY_BACKENDS} - \text{AV_LAUNCHER_PROCS} - \min(\max(\text{newValue}/4, 64), 1024)$. The values of [job_queue_processes](#), [autovacuum_max_workers](#), and [max_inner_tool_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 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 18-3 Value ranges in independent deployment mode

Memory Specifications	Value Range in CN	Value Range in DN
< 32GB	[10, 100]	[10, 100]
[32GB, 64GB)	[10, 200]	[10, 200]

Memory Specifications	Value Range in CN	Value Range in DN
[64GB, 128GB)	[10, 1000]	[10, 2500]
[128GB, 256GB)	[10, 2000]	[10, 6000]
[256GB, 480GB)	[10, 4000]	[10, 12000]
>= 480GB	[10, 8000]	[10, 24000]

Table 18-4 Value ranges in the finance edition (standard)

Memory Specificatio	Value Range in CN	Value Range in DN
< 64GB	[10, 100]	[10, 100]
[64GB, 128GB)	[10, 200]	[10, 1000]
[128GB, 256GB)	[10, 500]	[10, 2000]
[256GB, 480GB)	[10, 1000]	[10, 4000]
[480GB, 512GB)	[10, 2250]	[10, 9000]
[512GB, 576GB)	[10, 2500]	[10, 11000]
[576GB, 640GB)	[10, 3000]	[10, 12000]
[640GB, 768GB)	[10, 3500]	[10, 14000]
[768GB, 1024GB)	[10, 4000]	[10, 16000]
[1024GB, 1536GB)	[10, 6000]	[10, 21000]
>= 1536GB	[10, 10000]	[10, 33000]

Table 18-5 Value ranges in enterprise edition

Memory Specifications	Value Range in CN	Value Range in DN
< 64GB	[10, 100]	[10, 100]
[64GB, 128GB)	[10, 200]	[10, 900]
[128GB, 256GB)	[10, 350]	[10, 1500]
[256GB, 480GB)	[10, 900]	[10, 3500]
[480GB, 512GB)	[10, 1800]	[10, 7000]
[512GB, 576GB)	[10, 2000]	[10, 7500]
[576GB, 640GB)	[10, 2000]	[10, 8500]
[640GB, 768GB)	[10, 2500]	[10, 10000]

Memory Specifications	Value Range in CN	Value Range in DN
[768GB, 1024GB)	[10, 3000]	[10, 11000]
[1024GB, 1536GB)	[10, 4000]	[10, 15000]
>= 1536GB	[10, 7500]	[10, 24000]

Table 18-6 Value ranges in the finance edition (data computing)

Memory Specifications	Value Range in CN	Value Range in DN
< 256GB	[10, 100]	[10, 100]
[256GB, 512GB)	[10, 200]	[10, 1000]
[512GB, 576GB)	[10, 500]	[10, 4000]
[576GB, 768GB)	[10, 1000]	[10, 5000]
[768GB, 1024GB)	[10, 2500]	[10, 8000]
[1024GB, 1536GB)	[10, 4000]	[10, 12000]
>= 1536GB	[10, 5000]	[10, 18000]

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)
- Finance edition (standard):
CN: 6000 (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, and 96-core CPU/1024-GB memory); 4000 (96-core CPU/768-GB memory); 3500 (80-core CPU/640-GB memory); 3000 (72-core CPU/576-GB memory); 2500 (64-core CPU/512-GB memory); 2250 (60-core CPU/480-GB memory); 1000 (32-core CPU/256-GB memory); 500 (16-core CPU/128-GB memory); 200 (8-core CPU/64-GB memory)
DN: 21000 (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, and 96-core CPU/1024-GB memory); 16000 (96-core CPU/768-GB memory); 14000 (80-core CPU/640-GB memory); 12000 (72-core CPU/576-GB memory); 11000 (64-core CPU/512-GB memory); 9000 (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)
- Enterprise edition:
CN: 4000 (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory); 3000 (96-core CPU/768-GB memory); 2500

(80-core CPU/640-GB memory); 2000 (80-core CPU/512-GB memory, 72-core CPU/576-GB memory, and 64-core CPU/512-GB memory); 1800 (60-core CPU/480-GB memory); 900 (32-core CPU/256-GB memory); 350 (16-core CPU/128-GB memory); 200 (8-core CPU/64-GB memory)

DN: 15000 (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, and 96-core CPU/1024-GB memory); 11000 (96-core CPU/768-GB memory); 10000 (80-core CPU/640-GB memory); 8500 (72-core CPU/576-GB memory); 7500 (80-core CPU/512-GB memory and 64-core CPU/512-GB memory); 7000 (60-core CPU/480-GB memory); 3500 (32-core CPU/256-GB memory); 1500 (16-core CPU/128-GB memory); 900 (8-core CPU/64-GB memory)

- Finance edition (data computing):

CN: 2500 (96-core CPU/768-GB memory); 1000 (72-core CPU/576-GB memory); 500 (64-core CPU/512-GB memory); 200 (32-core CPU/256-GB memory)

DN: 8000 (96-core CPU/768-GB memory); 5000 (72-core CPU/576-GB memory); 4000 (64-core CPU/512-GB memory); 1000 (32-core CPU/256-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 the value of **max_connections** is not increased based on the external egress specifications and the memory parameter is not adjusted in proportion, when the service pressure is high, the memory may be insufficient, and the error message "memory is temporarily unavailable" is displayed.

In the hybrid deployment scenario, if the value of **max_connections** on the CN is greater than the value of **max_connections** divided by the number of CNs on the DN and the total pressure on the client exceeds the value of **max_connections** on the DN, the CN may fail to connect to the DN, and the error message "pooler... Too many clients already" 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 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. 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 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: **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 primary port number + 1 to clear useless sessions.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 18-1](#).

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: empty

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 VxxxRxxxCxx 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 18-1](#).

Value range: an integer ranging from 0 to 100000

check_disconnect_query

Parameter description: Specifies whether to terminate the execution of statements on the GaussDB server after the client is disconnected abnormally (for example, socketTimeout is triggered by JDBC, rwtimeout is triggered by libpq and the connection is closed, or the client process is terminated during service running).

Parameter type: Boolean

Unit: none

Value range:

- **on:** indicates that the GaussDB server stops running the corresponding statements after the client is disconnected unexpectedly.
- **off:** indicates that the GaussDB server does not stop running the corresponding statements after the client is disconnected unexpectedly.

Default value: on

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in [Table 18-1](#).

Setting suggestion: Retain the default value.

CAUTION

If this parameter is set and an upgrade is required, check whether the target version supports this parameter before the upgrade. If this parameter is not supported, delete it from the configuration file first before proceeding with the upgrade.

18.3.3.2 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 1 to 600. The smallest unit is s.

Default value: 1 min

auth_iteration_count

Parameter description: Specifies the number of iterations during the generation of encryption information for authentication.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 using the SET SESSION AUTHORIZATION syntax.

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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. 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: **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 gspl to Connect to an Instance](#).

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 gspl to Connect to an Instance](#).

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 CN.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: server.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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

Default value: an empty string

krb_srvname

Parameter description: Specifies the Kerberos service name.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

Default value: postgres

krb_caseins_users

Parameter description: Specifies whether the Kerberos username is case-sensitive.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 after cluster is installed.
- **off** indicates that the password of the initial account does not need to be modified after cluster is installed.

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 a GaussDB account.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 the parameter.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer consisting of 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer consisting of 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 999

- 0 means no limit.
- An integer ranging from 1 to 999 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 999

- 0 means no limit.
- An integer ranging from 1 to 999 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 999

- 0 means no limit.
- An integer ranging from 1 to 999 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- An integer ranging from 1 to 999 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 999. The unit is day.

- **0** indicates that the reminder is disabled.
- An integer ranging from 1 to 999 indicates the number of days prior to password expiration that a user will receive a reminder.

Default value: 7

18.3.3.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

tcp_keepalives_idle

Parameter description: Specifies the interval for transmitting keepalive signals on an OS that supports the TCP_KEEPIIDLE socket option. If no keepalive signal is transmitted, the connection is in idle mode.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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: 1 min

tcp_keepalives_interval

Parameter description: Specifies the response time before retransmission on an OS that supports the TCP_KEEPIIDLE socket option.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: 0 to 180. The unit is s.

Default value: 30

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.
-

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 keepalive 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-7 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

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)
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2
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 protocol to set up a data channel. The parameter setting takes effect after you restart the cluster.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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_control_port

Parameter description: Specifies the TCP listening port used by the TCP proxy communications library.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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** parameter plus 3.)

comm_max_datanode

Parameter description: Specifies the maximum number of DNs supported by the TCP proxy communication library.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 communication 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 **query_dop**.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication library.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication library. When you use a 1GE NIC, a small value ranging from 20 KB to 40 KB is recommended.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication 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 communication library or SCTP communication library may deteriorate. (SCTP is no longer used in the current version.)

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication library on a DN.

NOTICE

If the memory used by the communication library is small, set this parameter to a small value. Otherwise, set it to a large value.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication library on a DN. This parameter is used to adaptively reserve memory used by the communication libraries.

NOTICE

If the memory used by the communication library is small, set this parameter to a small value. Otherwise, set it to a large value.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 communication library to a specified IP address when the client initiates a connection.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 communication library connection.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 communication 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the debug logs of the communication library are printed.
- **off** indicates that the debug logs of the communication library are not printed.

Default value: off

comm_ackchk_time

Parameter description: Specifies the duration after which the communication library server automatically triggers ACK when no data packet is received.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 20000. The unit is ms. **0** indicates that automatic ACK triggering is disabled.

Default value: 2000 (2s)

comm_timer_mode

Parameter description: Specifies whether to enable the timer mode of the TCP proxy communication 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the timer logs of the communication library are printed.
- **off** indicates that the timer logs of the communication library are not printed.

Default value: off

comm_stat_mode

Parameter description: Specifies whether to enable the statistics mode of the TCP proxy communication 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the statistics logs of the communication library are printed.
- **off** indicates that the statistics logs of the communication 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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

COMM_PARAM

Parameter description: Specifies whether to print the **session** parameter settings during node communication.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.4 Resource Consumption

18.3.4.1 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

- If the value of **max_process_memory** minus **shared_buffer** minus **cstore_buffers** minus metadata size 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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)

Finance edition (standard):

CN: 200 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 160 GB (96-core CPU/768 GB memory); 130 GB (80-core CPU/640 GB memory); 120 GB (72-core CPU/576 GB memory); 100 GB (64-core CPU/512 GB memory and 60-core CPU/480 GB memory); 50 GB (32-core CPU/256 GB memory); 20 GB (16-core CPU/128 GB memory); 10 GB (8-core CPU/64 GB memory)

DN: 350 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 260 GB (96-core CPU/768 GB memory); 220 GB (80-core CPU/640 GB memory); 200 GB (72-core CPU/576 GB memory); 180 GB (64-core CPU/512 GB memory); 160 GB (60-core CPU/480 GB memory); 80 GB (32-core CPU/256 GB memory); 40 GB (16-core CPU/128 GB memory); 20 GB (8-core CPU/64 GB memory)

Enterprise edition:

CN: 150 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); 110 GB (96-core CPU/768 GB memory); 90 GB (80-core CPU/640 GB memory); 80 GB (72-core CPU/576 GB memory); 75 GB (80-core

CPU/512 GB memory, 64-core CPU/512 GB memory); 70 GB (60-core CPU/480 GB memory); 35 GB (32-core CPU/256 GB memory); 15 GB (16-core CPU/128 GB memory); 9 GB (8-core CPU/64 GB memory)

DN: 250 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 190 GB (96-core CPU/768 GB memory); 160 GB (80-core CPU/640 GB memory); 140 GB (72-core CPU/576 GB memory); 125 GB (80-core CPU/512 GB memory and 64-core CPU/512 GB memory); 120 GB (60-core CPU/480 GB memory); 60 GB (32-core CPU/256 GB memory); 25 GB (16-core CPU/128 GB memory); 15 GB (8-core CPU/64 GB memory)

Finance edition (data computing)

CN: 100 GB (96-core CPU/768 GB memory); 60 GB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 20 GB (32-core CPU/256 GB memory)

DN: 150 GB (96-core CPU/768 GB memory); 110 GB (72-core CPU/576 GB memory); 100 GB (64-core CPU/512 GB memory); 40 GB (32-core CPU/256 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 DNs.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

- If **enable_global_plancache** is set to **on**, **local_syscache_threshold** does not take effect when it is set to a value less than 16 MB to ensure that GPC takes effect. The minimum value is 16 MB.
- If **enable_global_syscache** and **enable_thread_pool** are set to **on**, 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 x 1024 to 512 x 1024. The unit is KB.

Default value:

- Independent deployment: **16 MB**
- Finance edition (standard):
32 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, and 80-core CPU/640 GB memory); 16 MB (72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, and 8-core CPU/64 GB memory)
- Enterprise edition:
32 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, and 64-core CPU/512 GB memory); 16 MB (60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, and 8-core CPU/64 GB memory)
- Finance edition (data computing): 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 function of checking whether the number of memory contexts exceeds the specified limit is enabled. This parameter applies only to the DEBUG version.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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)

Finance edition (standard):

CN: 2 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 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)

DN: 140 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 100 GB (96-core CPU/768 GB memory); 80 GB (80-core CPU/640 GB memory and 72-core CPU/576 GB memory); 70 GB (64-core CPU/512 GB memory); 60 GB (60-core CPU/480 GB memory); 30 GB (32-core CPU/256 GB memory); 16 GB (16-core CPU/128 GB memory); 8 GB (8-core CPU/64 GB memory)

Enterprise edition:

CN: 2 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 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)

DN: 100 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 76 GB (96-core CPU/768 GB memory); 64 GB (80-core CPU/640 GB memory); 56 GB (72-core CPU/576 GB memory); 50 GB (80-core CPU/512 GB memory and 64-core CPU/512 GB memory); 48 GB (60-core CPU/480 GB memory); 24 GB (32-core CPU/256 GB memory); 10 GB (16-core CPU/128 GB memory); 6 GB (8-core CPU/64 GB memory)

Finance edition (data computing):

CN: 2 GB (96-core CPU/768 GB memory); 1 GB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 512 MB (32-core CPU/256 GB memory)

DN: 50 GB (96-core CPU/768 GB memory); 40 GB (72-core CPU/576 GB memory);
30 GB (64-core CPU/512 GB memory); 10 GB (32-core CPU/256 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: Specifies the memory size of a GaussDB segment-page metadata page.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 16 to 1073741823. The unit is 8 KB.

The value of **segment_buffers** must be an integer multiple of **BLCKSZ**. Currently, **BLCKSZ** is set to **8 KB**. That is, the value of **segment_buffers** must be an integer multiple of 8 KB. The minimum value changes according to **BLCKSZ**.

Default value: 8 MB

Setting suggestions:

- **segment_buffers** is used to cache the content of segment-paged headers, which is key metadata information. To improve performance, it is recommended that the segment headers of ordinary tables be cached in the buffer and not be replaced. You are advised to set this parameter based on the following formula: Number of tables (including indexes and toast tables) x Number of partitions x 3 + 128. This is because each table (partition) has some extra metadata segments. Generally, a table has three segments. At last, **+128** is added because segment-page tablespace management requires a certain number of buffers.
- If this parameter is set to a small value, it takes a long time to create a segment-page table for the first time. Therefore, you are advised to set this parameter to the recommended value.

bulk_write_ring_size

Parameter description: Specifies the size of a ring buffer used for parallel data import.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 DN nodes 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 equal to or greater than that on primary servers. Otherwise, queries will fail on the standby servers.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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)
- Finance edition (standard):
CN: 900 (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); 800 (96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 400 (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 200 (8-core CPU/64 GB memory)
DN: 4200 (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 2200 (32-core CPU/256 GB memory); 1200 (16-core CPU/128 GB memory); 800 (8-core CPU/64 GB memory)
- Enterprise edition:
CN: 900 (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); 800 (96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 400 (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 200 (8-core CPU/64 GB memory)
DN: 1800 (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); 1200 (96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 800 (32-core CPU/256 GB memory); 400 (16-core CPU/128 GB memory and 8-core CPU/64 GB memory)
- Finance edition (data computing):
CN: 800 (96-core CPU/768 GB memory); 400 (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 200 (32-core CPU/256 GB memory)
DN: 2400 (96-core CPU/768 GB memory); 1200 (72-core CPU/576 GB memory); 800 (64-core CPU/512 GB memory); 400 (32-core CPU/256 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 equal to or greater than 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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)
- Finance edition (standard):
CN: 128 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 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)
DN: 256 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, and 96-core CPU/768 GB memory); 128 MB (80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 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)
- Enterprise edition:
128 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 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)
- Finance edition (data computing):
128 MB (96-core CPU/768 GB memory, 72-core CPU/576 GB memory, and 64-core CPU/512 GB memory); 64 MB (32-core CPU/256 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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)
- Finance edition (standard):
CN: 1 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, and 80-core CPU/640 GB memory); 512 MB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 256 MB (60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, and 8-core CPU/64 GB memory)
DN: 2 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 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)
- Enterprise edition:
CN: 1 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, and 80-core CPU/512 GB memory); 512 MB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 256 MB (60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, and 8-core CPU/64 GB memory)

DN: 2 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 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)

- Finance edition (data computing):

CN: 1 GB (96-core CPU/768 GB memory); 256 MB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 128 MB (32-core CPU/256 GB memory)

DN: 2 GB (96-core CPU/768 GB memory); 1 GB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 512 MB (32-core CPU/256 GB memory)

NOTICE

Setting suggestions:

- The value of this parameter must be greater than that of [work_mem](#) so that database dumps can be more quickly cleared or restored. 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 [Automatic Vacuuming](#) threads are running, the memory equal to a multiple of the value of [autovacuum_max_workers](#) is allocated. In this case, set [maintenance_work_mem](#) to a value equal to or greater than 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 is a SUSE parameter. Set it based on instructions provided in [Table 18-1](#).

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 equal to or greater than 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 to query the recorded memory snapshot information. For details, see "gs_get_history_memory_detail(cstring)" in "SQL Reference > Functions and Operators" > "Statistics Information 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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.
-

18.3.4.2 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 is a USERSET parameter. Set it based on instructions provided in [GUC parameters](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.4.3 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 physical machine $\times$ **max_files_per_process** $\times$ 3

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

18.3.4.4 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:

- **vacuum_cost_delay** x **accumulated_balance/vacuum_cost_limit**
 - **vacuum_cost_delay** x 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. 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 100. A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

Default value:

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 1 to 10000

Default value: 1000

18.3.4.5 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 periodical 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 background writer 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 10 to 10000. The unit is ms.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 3600000. The unit is ms.

Default value: 2000ms(2s)

max_io_capacity

Parameter description: Specifies the I/O upper limit per second for the background writer 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 background writer process. If a large number of pages are evicted due to service triggering, the services are affected.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer, in the range [32,256]

Default value: 256

18.3.4.6 Asynchronous I/O

checkpoint_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the checkpoint 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 checkpoint 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

18.3.5 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 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 **on** and the problem "the imported file does not exist" occurs, GaussDB will report the error message "file does not exist".

This is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 restrictions are posed on COPY and advanced package paths. The path cannot end with a slash (/) or contain periods (..).

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string of up to 4096 characters

Default value: NULL

enable_copy_server_files

Parameter description: Specifies whether to enable the permission to copy server files.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 **SYSDADMIN** 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.

18.3.6 Write Ahead Log

18.3.6.1 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 the 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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.
- **fdasync** indicates that **fdasync()** is called at each commit. (SUSE 10 and SUSE 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**.

- **fsync** indicates that **fsync()** is called at each commit (SUSE 10 and SUSE 11 are supported).
- **open_sync** indicates that the **open()** with the O_SYNC option is used to write WAL files (SUSE 10 and SUSE 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: -1 to 2¹⁸. 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)
- Finance edition (standard):
CN: 512 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, and 80-core CPU/640 GB memory); 256 MB (72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 128 MB (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 64 MB (8-core CPU/64 GB memory)
DN: 2 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, and 96-core CPU/768 GB memory); 1 GB (80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 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)
- Enterprise edition:
CN: 512 MB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, and 80-core CPU/512 GB memory); 256 MB (72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 128 MB (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 64 MB (8-core CPU/64 GB memory)
DN: 1 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 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)
- Finance edition (data computing):
CN: 256 MB (96-core CPU/768 GB memory); 125 MB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 64 MB (32-core CPU/256 GB memory)
DN: 1 GB (96-core CPU/768 GB memory, 72-core CPU/576 GB memory, and 64-core CPU/512 GB memory); 512 MB (32-core CPU/256 GB memory)

Setting suggestions: The content of the WAL buffers is written to disks at every transaction commit, so extremely large values are unlikely to provide a significant increase in system performance. However, setting this parameter to a few megabytes can improve the disk write performance on a server to which a large number of transactions are immediately committed. 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

Default value: false

walwriter_sleep_threshold

Parameter description: Specifies the number of times that the idle Xlog is refreshed before the Xlog refresher enters sleep.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 (microseconds)

Default value: 2

wal_flush_delay

Parameter description: Specifies the wait interval when an entry in the **WAL_NOT_COPIED** state is encountered during **WalInsertStatusEntryTbl** traversal.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 0 to 90000000 (microseconds)

Default value: 1

18.3.6.2 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 30 to 3600. The unit is s.

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: 15 min

checkpoint_completion_target

Parameter description: Specifies the target of checkpoint completion.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647. The unit is s. **0** indicates that the warning is disabled.

Default value: 5 min

Recommended value: 5 min

checkpoint_wait_timeout

Parameter description: Sets the longest time that the checkpoint waits for the checkpoint thread to start.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 2 to 3600. The unit is s.

Default value: 1 min

enable_incremental_checkpoint

Parameter description: Specifies whether to enable incremental checkpoint.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 1 to 3600. The unit is s.

Default value: 1 min

enable_xlog_prune

Parameter description: Specifies whether the primary node recycles logs if the size of Xlogs exceeds the value of **max_size_for_xlog_prune** when any standby node is disconnected.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- If this parameter is set to **on**, the primary node recycles logs when any standby node is disconnected.
- If this parameter is set to **off**, the primary node does not recycle 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 replay 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 163840 to 2147483647. The unit is KB.

Default value: 1048576. The unit is KB.

18.3.6.3 Log Replay

recovery_time_target

Parameter description: Specifies the time for a standby server to write and replay logs.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 server can write and replay logs within the period specified by the value, so that the standby server 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

recovery_parse_workers

Parameter description: Specifies the number of **ParseRedoRecord** threads for the ultimate RTO feature.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 setting of **recovery_parse_workers** prevails and the concurrent replay function is disabled. Due to the ultimate RTO configuration requirements, the value of **recovery_parse_workers** can be greater than 1 only when **replication_type** is set to 1. In addition, when ultimate RTO is enabled, ensure that the value of **wal_receiver_buffer_size** is equal to or greater than 32 MB (64 MB is recommended).

Default value: 1

 NOTE

- After V500R001C00 is upgraded 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 will start more worker threads whose value is equal to the value of **recovery_parse_workers** x (the value of **recovery_redo_workers** + 2) + 5, occupying more CPU, memory, and I/O resources. In hybrid deployment scenarios, host performance may be affected.
- 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 1 to 8

This parameter must be used together with **recovery_parse_workers**, and it takes effect only when **recovery_parse_workers** is set to a value greater than 1.

Default value: 1

 NOTE

After V500R001C00 is upgraded 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. If the number of CPUs is less than 16, you are advised to set this parameter to **2**. If the number is greater than 16 and less than 32, you are advised to set this parameter to **4**. If the number is greater than 32, you are advised to set this parameter to **8**.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

Default value: on

redo_bind_cpu_attr

Parameter description: Specifies the core binding operation of the replayer thread. Only the **sysadmin** user can access this parameter. This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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. In hybrid deployment scenarios, you are advised to bind the replay threads of different nodes on the same host to different NUMA groups.

18.3.6.4 Archiving

archive_timeout

Parameter description: Specifies the archiving period.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 s.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.7 HA Replication

18.3.7.1 Sending Server

max_wal_senders

Parameter description: Specifies the maximum number of simultaneously running WAL sender processes. The value cannot be equal to or greater than that of [max_connections](#).

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

NOTICE

[wal_level](#) must be set to **archive**, **hot_standby**, or **logical** to allow the connection 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 equal to or greater than 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 WAL synchronization is proper.

wal_sender_timeout

Parameter description: Specifies the maximum duration that the sending server waits for the WAL receiving in the receiver.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 a value by referring to the following formula: Number of current physical streaming 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 streaming replication slots required by the cluster is as follows: ratio of the number of standby and secondary DNs to the number of primary DNs in a ring of DNs. For example, if the HA cluster has one primary DN and two standby DNs, the number of physical replication slots required is 2. If the HA cluster has 1 primary DN and 3 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 remoteport=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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 remoteport=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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

18.3.7.2 Primary Server**synchronous_standby_names**

Parameter description: Specifies a comma-separated list of names of potential standby servers that support synchronous replication.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

- *num_sync* indicates the number of synchronous standby nodes that need to wait for responses before the transaction is committed, *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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 the primary and standby nodes.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

NOTICE

- This parameter is used for testing performance with standby DN and without standby DN. After this parameter is set to **off**, tests for abnormal scenarios such as switchovers and fault rectifications cannot be performed.
 - This parameter is a restricted 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 between primary and standby nodes is enabled.
- **off** indicates that data and log synchronization between the primary and standby nodes is disabled.

Default value: on

enable_mix_replication

Parameter description: Specifies how WAL files and data are replicated between primary and standby nodes.

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, so that VACUUM and VACUUM FULL do not clean up deleted tuples immediately.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

hadr_recovery_time_target

Parameter description: Specifies whether the standby database instance completes log writing and replay in streaming DR mode.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 **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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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.

18.3.7.3 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

NOTICE

-1 indicates that the standby server 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 a standby server are canceled when the queries conflict with WAL data receiving through streaming replication in hot standby mode.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

NOTICE

-1 indicates that the standby server 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 nodes to store the received Xlog files.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 replay is enabled.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** indicates that atomic operations are used for update.
- **off** indicates that spinlocks are used for update.

Default value: on

18.3.8 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.

18.3.8.1 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** parameter, or 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on:** enabled.
- **off:** disabled.

Default value: on

enable_material

Parameter description: Specifies the query optimizer's use of materialization.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on:** enabled.
- **off:** disabled.

Default value: off

enable_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to fully scan internal tables. If this variable is disabled, the optimizer preferentially selects another method when another method exists.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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. If this variable is disabled, the optimizer preferentially selects another method when another method exists.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on:** enabled.
- **off:** disabled.

Default value: on

enable_sort

Parameter description: Specifies the query optimizer's choice of sort methods. If this variable is disabled, the optimizer preferentially selects another method when another method exists.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on:** enabled.
- **off:** disabled.

Default value:

- Independent deployment: **off**
- Finance edition (standard): **on**
- Enterprise edition: **on**
- Finance edition (data computing): **on**

enable_stream_recursive

Parameter description: Specifies whether to push **WITH RECURSIVE** join queries to DN for processing.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647

Default value: 200

enable_broadcast

Parameter description: Controls the query optimizer's use of broadcast distribution method when it evaluates the cost of stream.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 the columns is the distribution key, a **group by** column will be selected for redistribution. This parameter specifies the policy of selecting a redistribution key.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 computing node groups are listed for the optimizer to select the most appropriate one for the current operator.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 list elements in the IN clause 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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**

cost_weight_index

Parameter description: Specifies the cost weight of index_scan.

- If this parameter is set to **1**, no adjustment is performed.
- If this parameter is set to a value less than 1, the cost of `index_scan` is reduced and `index_scan` is more likely to be selected by the optimizer.
- If this parameter is set to a value greater than 1, the cost of `index_scan` increases.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 a generic plan. If this parameter is set to a positive value, the positive value is used as the estimated number of limit rows. If the positive value is a decimal, the value is rounded up automatically. 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the A matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: on

check_implicit_conversions

Parameter description: Specifies whether to check that candidate index paths are generated for index columns with implicit type conversion in a query. For details about the application scenarios of this parameter, see "SQL Optimization > Checking the Implicit Conversion Performance" in *Developer Guide*.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the system checks whether candidate index paths are generated for index columns with 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.

18.3.8.2 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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)

Finance edition (standard):

CN: 1 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 512 MB (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 256 MB (8-core CPU/64 GB memory)

DN: 70 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 50 GB (96-core CPU/768 GB memory); 40 GB (80-core CPU/640 GB memory and 72-core CPU/576 GB memory); 35 GB (64-core CPU/512 GB memory); 30 GB (60-core CPU/480 GB memory); 15 GB (32-core CPU/256 GB memory); 8 GB (16-core CPU/128 GB memory); 4 GB (8-core CPU/64 GB memory)

Enterprise edition:

CN: 1 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 512 MB (32-core CPU/256 GB memory and 16-core CPU/128 GB memory); 256 MB (8-core CPU/64 GB memory)

DN: 50 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 38 GB (96-core CPU/768 GB memory); 32 GB (80-core CPU/640 GB memory); 28 GB (72-core CPU/576 GB memory); 25 GB (80-core CPU/512 GB memory and 64-core CPU/512 GB memory); 24 GB (60-core CPU/480 GB memory); 12 GB (32-core CPU/256 GB memory); 5 GB (16-core CPU/128 GB memory); 3 GB (8-core CPU/64 GB memory)

Finance edition (data computing):

CN: 1 GB (96-core CPU/768 GB memory); 512 MB (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 256 MB (32-core CPU/256 GB memory)

DN: 25 GB (96-core CPU/768 GB memory); 20 GB (72-core CPU/576 GB memory); 15 GB (64-core CPU/512 GB memory); 5 GB (32-core CPU/256 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: 0

18.3.8.3 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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:** enabled.
- **off:** 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

18.3.8.4 Other Optimizer Options

enable_fast_query_shipping

Parameter description: Specifies whether to use the distributed framework for a query optimizer.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the distributed framework is not used. Execution plans are generated on CNs and DN separately.
- **off** indicates that the distributed framework is used. Execution plans are generated on CNs and then sent to DNs for execution.

Default value: on

enable_trigger_shipping

Parameter description: Specifies whether the trigger can be pushed to DNs for DN execution.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the trigger can be pushed to DNs for DN 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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_remotesort

Parameter description: Specifies whether the execution plan of the **ORDER BY** clause can be delivered to DNs for execution.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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 optional query rewriting rules that have been enabled. 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 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

- **none:** No optional query rewriting rules are used.
- **lazyagg:** The Lazy Agg query rewriting rules are used to eliminate 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:** The Partial Push query rewriting rules are used. For statements that cannot be pushed down, push down some subqueries to DN for execution and the rest to CN for execution.
- **uniquecheck:** The Unique Check query rewriting rules are used to optimize the subquery statements in target columns without agg and check whether the number of returned rows is 1.
- **disablerep:** The Disable Replicate query rewriting rules are used. The performance may deteriorate after a replication table is optimized. Therefore, after this rule is enabled, subqueries cannot be optimized.
- **intargetlist:** The In Target List query rewriting rules are used to improve subqueries in the target column.
- **predpushnormal:** The Predicate Push query rewriting rules are used. When predicate conditions are pushed down to subqueries, the BROADCAST operator may be added to support distributed execution.
- **predpushforce:** The Predicate Push query rewriting rules are used to push down predicate conditions to subqueries and use indexes as much as possible for acceleration.
- **predpush:** The optimal plan is selected 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*.
- **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*.
- **enable_sublink_pullup_rownum:** The optimizer can be used to promote subjoins when the SQL statement contains the ROWNUM pseudocolumn.

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the optimizer optimizes the query plan.
- **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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 of the PBE query execution plan. 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

When **enable_global_plancache** is enabled, to ensure that GPC takes effect, the value of **local_syscache_threshold** must be equal to or greater than 16 MB. 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 the execution plan of the PBE query is shared in the cache.
- **off** indicates that the execution plan of the PBE query is not shared in the cache.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- on
- off

Default value: off

enable_opfusion

Parameter description: Specifies whether to optimize simple queries.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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**: enabled.
- **off**: disabled.

Default value: on

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

- **none:** None of the beta optimizer features is 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:** advances the canonical pathkey generation (**pathkey:** a set of ordered key values of data).
- **index_cost_with_leaf_pages_only:** Considers index leaf nodes when the index cost is estimated.
- **a_style_coerce:** enables the Decode type conversion rule to be compatible with ORA. For details, see case processing in O-compatible 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.
- **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.

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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**. The number of rows sampled during statistics collection is affected.

If this parameter is set to a positive number, the number of rows sampled in the statistics histogram is **default_statistics_target** x 300. 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%. The number of sampled rows is the total number of rows multiplied by 5%. This parameter affects only the target number of sampled rows in the statistics. The actual number of sampled rows is also affected by the memory parameter [work_mem](#).

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 Optimization" in *Developer Guide*.
- If this parameter is set to a negative value, the number of samples is equal to or greater than 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 when this parameter uses its default value.
- If this parameter is set to a negative value, the auto-analyze function is disabled.

Default value: 100

constraint_exclusion

Parameter description: Controls the query optimizer's use of table constraints to optimize queries.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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. If this parameter is set to **on** for all tables, extra planning is imposed on simple queries, which has no benefits. 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 JOIN**) into lists of **FROM** items based on the number of the items in the result list.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 value from **1** to **2147483647** is regarded as the seed identifier, based on which the optimizer generates a random execution plan.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from -1 to 2147483647

Default value: 0

NOTICE

- If **plan_mode_seed** 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 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 DN for random retrieval to release the pressure on nodes.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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 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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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

NOTE

This parameter is irrelevant to autoanalyze in the autovacuum thread.

query_dop

Parameter description: Specifies the user-defined degree of parallelism (DOP). This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

- When the hash aggregation operator designed for column-oriented hash tables is used, the memory usage of the query 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_hashagg** is enabled and the hash aggregation 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 Guide" > "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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 Guide > 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is forcibly used (soft parse). **generic plan** is a plan generated after you run the **prepare** statement. The plan policy binds parameters to the plan when you run the **execute** statement and execute the plan. The advantage of this scheme 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 is forcibly used (hard parse). **custom plan** is a plan generated after you run the **prepare** statement where parameters in the **execute** statement is embedded. The **custom plan** generates a plan based on specific parameters in the **execute** statement. This scheme 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 the **prepare** statement. It is used when the parameterized field in the **prepare** statement has severe data skew.

Default value: auto

enable_router

Parameter description: Specifies whether to enable the manual node pushdown function.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on:** enabled.
- **off:** disabled.

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 how long execution plans are automatically printed for. Plans can be printed only when the time required to execute the plans is greater than the value of **auto_explain_log_min_duration**.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

- **0**: All executed plans are generated.
- **3000**: All execution plans will be generated after 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 prepare precompilation mode can be displayed. Plans generated in JDBC precompilation mode cannot be displayed.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 8192

Default value: 0

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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.

18.3.9 Error Reporting and Logging

18.3.9.1 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** can be set to "**stderr, csvlog**".)

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

The valid values are **stderr**, **csvlog**, **syslog**, and **eventlog**.

- **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.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: PostgreSQL

18.3.9.2 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 18-8](#). 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

NOTICE

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 18-8](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-8](#).

 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 equal to or greater than 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

NOTICE

When using this option 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 equal to or greater than the level specified by this parameter.

This is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-8](#).

Default value: **panic**

[Table 18-8](#) 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 18-8 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 VACUUM VERBOSE.	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, COMMIT 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

18.3.9.3 Logging Content

debug_print_parse

Parameter description: Specifies whether to print parsing tree results.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-1](#).

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 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-9](#).

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Table 18-9 Escape characters

Escape Character	Effect
%a	Application name
%u	Username
%d	Database name
%r	Remote host name or IP address and remote port. If log_hostname is set to off , only the IP address and remote port are displayed.
%h	Remote host name or IP address. If log_hostname is set to off , 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 (0 indicates that no transaction ID is assigned)
%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

Escape Character	Effect
%%	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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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.

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

Default value: **off**. All the module logs are not output on the server. You can view the logs by running **show logging_module**.

```
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCR_CHKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
```

Setting method: Run **show logging_module** to view which modules are controllable. For example, the query output result is as follows:

```
openGauss=# show logging_module;
logging_module
-----
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCR_CHKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
(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:

```
openGauss=# set logging_module='on(SSL)';
SET
openGauss=# show
logging_module;
      logging_module
-----
ALL,on(SSL),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCR_CHKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
(1 row)
```

SSL log output is enabled.

The **ALL** identifier can be used to quickly enable or disable log output for all modules.

```
openGauss=# set logging_module='off(ALL)';
SET
openGauss=# show
```



```
logging_module;  
logging_module  
-----  
-----  
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECU  
TOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANA  
LYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQ  
L,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,CO  
MM_PARAM,ENCODING_CHECK)  
(1 row)  
  
openGauss=# set logging_module='on(ALL)';  
SET  
openGauss=# show  
logging_module;  
logging_module  
-----  
-----  
ALL,on(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,  
STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE  
,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TE  
XTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_  
PARAM,ENCODING_CHECK),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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 amount 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.9.4 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 18-10](#).

Table 18-10 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
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

Field	Description	Field	Description
transaction_id	Transaction ID	location	Position where errors occur in the GaussDB source code if log_error_verbosity is set to verbose
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.

18.3.10 Alarm Detection

During the running of the cluster, database cluster errors 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 DN.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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`

18.3.11 Statistics During the Database Running

18.3.11.1 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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** 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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.
- **all** indicates that SQL statements 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 100 to 102400

Default value: 1024

track_procedure_sql

Parameter description: Specifies whether the SQL statements that are being executed in the stored procedure are printed in the query column in the **pg_stat_activity** system catalog.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on:** indicates that when a stored procedure is called, the statements that are being executed by the stored procedure are printed in the query column of **pg_stat_activity**.
- **off:** indicates that when a stored procedure is called, only statements for invoking the stored procedure are printed in the query column of **pg_stat_activity**.

Default value: on

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 is an INTERNAL parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 1440. The unit is min.

Default value: 30 min

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

track_sql_count

Parameter description: Collects statistics on the statements (**SELECT**, **INSERT**, **UPDATE**, **MERGE INTO**, and **DELETE**) that are being executed in a session.

This is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: on

 NOTE

If **track_sql_count** is set to **off**, querying the **gs_sql_count** or **pgxc_sql_count** view returns 0.

18.3.11.2 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 are SUSE parameters. Set them based on instructions in [Table 18-1](#).

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

18.3.11.3 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 statistics collection function is disabled.

Default value: off

18.3.12 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

NOTE

- Set the **autovacuum** parameter to **on** if you want to start the automatic vacuuming of abnormal two-phase transactions when the system recovers from faults.
- If **autovacuum** is set to **on** and **autovacuum_max_workers** to **0**, the autovacuum process is started only when the system recovers from faults to vacuum 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.

NOTICE

Even if **autovacuum** is set to **off**, the autovacuum process will be started automatically when a transaction ID wraparound is about to occur. When a **CREATE DATABASE** or **DROP DATABASE** operation fails, it is possible that the transaction has been committed or rolled back on some nodes whereas some nodes are still in the prepared state. In this case, perform the following operations to manually restore the nodes:

1. Use the `gs_clean` tool (setting the **option** parameter to -N) to query the xid of the abnormal two-phase transaction and nodes in the prepared status.
2. Log in to the nodes in the prepared state. Administrators connect to an available database such as **postgres** and run the **set xc_maintenance_mode = on** statement.
3. Commit or roll back the two-phase transaction based on the global transaction status.

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 started. This parameter is valid only when **autovacuum** is set to **on**.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 equal to or greater than 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 greater than 0 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 18-2](#).

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 - Value of max_inner_tool_connections - Value of max_connections - Value of job_queue_processes - Number of auxiliary threads - 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 1 to 2147483. The unit is s.

Default value: 10 min (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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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.relFrozenxid` field 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 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from -1 to 100. The unit is ms. -1 indicates that the normal vacuum cost delay is used.

Default value: 20ms

autovacuum_vacuum_cost_limit

Parameter description: sets the value of the cost limit used in the **autovacuum** operation.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from -1 to 10000 -1 indicates that the normal 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 1 to 10

Default value: 3

defer_csn_cleanup_time

Parameter description: Specifies the interval of recycling transaction IDs.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 0 to 2147483647

Default value: 5s (5000 ms)

18.3.13 Default Settings of Client Connection

18.3.13.1 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

- If the schema of temporary tables exists in the current session, the schema can be listed in the 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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.
- If you need to obtain a schema during GaussDB development, use the value of **search_path** because the schema is determined by **search_path**. To be compatible with database ORA, **current_schema** is used only to modify the value of **search_path**.

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

- **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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

CAUTION

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#). 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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: 2000000000

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.relFrozenxid64](#) is greater than the specified time.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 1 to 2147483647

Default value: 1000

18.3.13.2 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 18-1](#).

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 18-1](#).

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**: Output matching 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 18-1](#).

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 18-1](#).

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 18-1](#).

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 18-1](#).

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 SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-1](#).

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 18-1](#).

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 18-1](#).

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

18.3.13.3 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 18-1](#).

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

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 username 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 is a BACKEND parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

Default value: empty

18.3.14 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 ordinary 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 is a SUSE parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSERSET parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: 120000 (2 minutes)

max_locks_per_transaction

Parameter description: Determines the average number of object locks allocated to each transaction.

Parameter type: integer

Unit: none

Value range: 10 to 2147483647

Default value: 256

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in [Table 18-1](#).

Setting suggestions:

- The maximum number of hash tables that can be locked by a shared lock at any time is calculated based on an assumption: $\text{max_locks_per_transaction} \times (\text{max_connections} + \text{max_prepared_transactions})$ and $\text{max_locks_per_transaction} \geq \text{Number of concurrent service transactions} \times \text{Number of object locks added by each service transaction} / (\text{max_connections} + \text{max_prepared_transactions})$. In this case an upper limit is determined for this parameter.
- Within the specified range, objects can be locked simultaneously at any time. You may need to increase the value of this parameter if many different tables are modified in a single transaction.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows, leading to database startup failure.

- When running a standby node, you must set this parameter to a value that is no less than that on the primary node. Otherwise, queries will not be allowed on the standby node.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 call the **gs_clean** tool to clean unfinished two-phase transactions in cluster. **gs_clean_timeout** is used to control the interval for the Coordinator to call the **gs_clean** tool.
- A larger value of this parameter indicates a low frequency of calling **gs_clean** in GaussDB to clean unfinished two-phase transactions.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 1440. The unit is minute.

Default value: 5min

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 1 to 1000

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 ordinary 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, that is, the number of lock partitions is 16. The maximum value is 16, that is, 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 equal to or greater than 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value:

- **CLOG_PART:** 256
- **CSNLOG_PART:** 512
- **LOG2_LOCKTABLE_PART:** 4

- **TWOPHASE_PART**: 1
- **FASTPATH_PART**: 20

18.3.15 Version and Platform Compatibility

18.3.15.1 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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'...').

Value range: 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSER parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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

18.3.15.2 Platform and Client Compatibility

Many platforms use the database system. External compatibility of the database system provides a lot of 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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

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.
 - 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: ""

NOTE

- Currently, only [Table 18-11](#) is supported.
- Multiple items are separated by commas (,), for example, **set behavior_compat_options='end_month_calculate,display_leading_zero'**;

Table 18-11 Compatibility configuration items

Compatibility Configuration Item	Compatibility Behavior Control
display_leading_zero	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. - If this item is not specified, for a non-zero decimal number between -1 and +1, the 0 before the decimal point is not displayed. For example: openGauss=# 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) - If this item is specified, for a non-zero decimal number between -1 and +1, the 0 before the decimal point is displayed. For example: openGauss=# 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)

Compatibility Configuration Item	Compatibility Behavior Control
end_month_calculate	Specifies the calculation logic of the add_months function. Assume that the two parameters of the add_months function are param1 and param2, and that the month of param1 and param2 is result. • If this item is not specified, and the Day of param1 indicates the last day of a month shorter than result, the Day in the calculation result will equal that in param1. For example: <pre>openGauss=# select add_months('2018-02-28',3) from sys_dummy; add_months ----- 2018-05-28 00:00:00 (1 row)</pre> • If this item is specified, and the Day of param1 indicates the last day of a month shorter than result, the Day in the calculation result will equal that in result. For example: <pre>openGauss=# select add_months('2018-02-28',3) from sys_dummy; add_months ----- 2018-05-31 00:00:00 (1 row)</pre>
compat_analyze_sample	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.
bind_schema_tablespace	Binds a schema with the tablespace with the same name. If a tablespace name is the same as <i>sche_name</i>, default_tablespace will also be set to <i>sche_name</i> if search_path is set to <i>sche_name</i>.
bind_procedure_searchpath	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: • If this item is not specified, the system reports an error and exits. • If this item is specified, the search continues based on the settings of search_path. If the issue persists, the system reports an error and exits.

Compatibility Configuration Item	Compatibility Behavior Control
correct_to_number	Controls the compatibility of the to_number() 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. <pre>openGauss=# select " AS to_number_14, to_number('34,50','999,99'); ERROR: invalid data. CONTEXT: referenced column: to_number</pre> If this item is set, the result of the to_number() function is the same as that of pg11. <pre>openGauss=# select " AS to_number_14, to_number('34,50','999,99'); to_number_14 to_number -----+----- 3450 (1 row)</pre>
unbind_divide_bound	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 INT_MIN/(-1) is greater than INT_MAX. <pre>openGauss=# select (-2147483648)::int4 / (-1)::int4; ERROR: integer out of range</pre> If this item is specified, the range of the division result does not need to be verified. For example, the output result of INT_MIN/(-1) is INT_MAX+1. <pre>openGauss=# select (-2147483648)::int4 / (-1)::int4; ?column? ----- 2147483648 (1 row)</pre>
convert_string_digit_to_numeric	Determines whether to convert numeric constants of the character string type to those of the numeric type before these two types are compared. <pre>openGauss=# create table test1 (c1 int, c2 varchar); openGauss=# insert into test1 values (2, '1.1'); openGauss=# set behavior_compat_options=""; openGauss=# select * from test1 where c2 > 1; ERROR: invalid input syntax for type bigint: "1.1"</pre> openGauss=# set behavior_compat_options='convert_string_digit_to_numeric'; openGauss=# select * from test1 where c2 > 1; <pre>c1 c2 ----+----- 2 1.1 (1 row)</pre>

Compatibility Configuration Item	Compatibility Behavior Control
return_null_string	Specifies how to display the empty result (empty string '') of the lpad() and rpad() functions. - If this item is not specified, the empty string is displayed as NULL. <pre>openGauss=# select length(lpad('123',0,'*')) from sys_dummy; length ----- (1 row)</pre> - If this item is specified, the empty string is displayed as single quotation marks (''). <pre>openGauss=# select length(lpad('123',0,'*')) from sys_dummy; length ----- 0 (1 row)</pre>
compat_concat_variadic	Specifies the compatibility of variadic results of the concat() and concat_ws() functions. The MY database does not have the variadic type. Therefore, this option has no impact on the MY database. 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. <pre>openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ----- t (1 row)</pre> 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. -- In the ORA database: <pre>openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ----- t (1 row)</pre> -- In the TD database: <pre>openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ----- f (1 row)</pre>

Compatibility Configuration Item	Compatibility Behavior Control
merge_update_multi	When MERGE INTO... When WHEN MATCHED THEN UPDATE (see "SQL Reference" > "SQL Syntax" > "MERGE INTO" in <i>Developer Guide</i>) and INSERT ... ON DUPLICATE KEY UPDATE (see "SQL Reference" > "SQL Syntax" > "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. 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.
plstmt_implicit_savepoint	Determines whether the execution of an UPDATE statement in a stored procedure has an independent subtransaction. 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 EXCEPTION behavior of the O database.
hide_tailing_zero	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. For example: <pre>openGauss=# set behavior_compat_options='hide_tailing_zero'; openGauss=# 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) openGauss=# set behavior_compat_options=''; openGauss=# 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	After this parameter is enabled, the definer permission is used by default when a stored procedure is created.

Compatibility Configuration Item	Compatibility Behavior Control
char_coerce_compat	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. After 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 sql_compatibility parameter is set to ORA. After this parameter is enabled, spaces at the end are not omitted in implicit conversion, explicit conversion, or conversion by calling the text(bpchar) function. <pre> openGauss=# set behavior_compat_options=""; openGauss=# create table tab_1(col1 varchar(3)); openGauss=# create table tab_2(col2 char(3)); openGauss=# insert into tab_2 values(' '); openGauss=# insert into tab_1 select col2 from tab_2; openGauss=# select * from tab_1 where col1 is null; col1 ----- (1 row) openGauss=# select * from tab_1 where col1=' '; col1 ----- (0 rows) openGauss=# delete from tab_1; openGauss=# set behavior_compat_options = 'char_coerce_compat'; openGauss=# insert into tab_1 select col2 from tab_2; openGauss=# select * from tab_1 where col1 is null; col1 ----- (0 rows) openGauss=# select * from tab_1 where col1=' '; col1 ----- (1 row) </pre>
truncate_numeric_tail_zero	Configuration item for numeric display. If this parameter is not specified, numeric values are displayed based on the default precision. If this parameter is specified, the last 0 after the decimal point is hidden in all numeric output scenarios except to_char(numeric, format). For example: <pre> openGauss=#set behavior_compat_options='truncate_numeric_tail_zero'; openGauss=#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) openGauss=#set behavior_compat_options=""; openGauss=#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	Compatibility Behavior Control
array_count_com pat	Controls the array.count function. If the parameter is enabled, the function returns 0 . Otherwise, the function returns null.
aformat_regexp_ match	Determines the matching behavior of regular expression functions. When this parameter is set and sql_compatibility is set to ORA or MYSQL, the options supported by the flags parameter of the regular expression are changed as follows: 1. By default, the character '\n' cannot be matched. 2. When flags contains the n option, the character '\n' can be matched. 3. The regexp_replace(source, pattern replacement) function replaces all matching substrings. 4. regexp_replace(source, pattern, replacement, flags) returns null when the value of flags is '' or null. Otherwise, the meanings of the options supported by the flags parameter of the regular expression are as follows: 1. By default, the character '\n' can be matched. 2. The n option in flags indicates that the multi-line matching mode is used. 3. The regexp_replace(source, pattern replacement) function replaces only the first matched substring. 4. If the value of flags is '' or null, the return value of regexp_replace(source, pattern, replacement, flags) is the character string after replacement.
disable_emptystr 2null	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, byteawithoutorderwithqual-col, and byteawithoutordercol. This parameter is reserved for emergency. Do not set it unless necessary.

Compatibility Configuration Item	Compatibility Behavior Control
proc_uncheck_default_param	When a function is called, the system does not check whether the default parameter is omitted. - 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> openGauss=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int openGauss-# as openGauss\$\$ begin openGauss\$\$ raise info 'f1:%',f1; openGauss\$\$ raise info 'f2:%',f2; openGauss\$\$ raise info 'f3:%',f3; openGauss\$\$ raise info 'f4:%',f4; openGauss\$\$ raise info 'f5:%',f5; openGauss\$\$ return 1; openGauss\$\$ end; openGauss\$\$ / CREATE FUNCTION openGauss=# 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> - 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> openGauss=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int openGauss-# as openGauss\$\$ begin openGauss\$\$ raise info 'f1:%',f1; openGauss\$\$ raise info 'f2:%',f2; openGauss\$\$ raise info 'f3:%',f3; openGauss\$\$ raise info 'f4:%',f4; openGauss\$\$ raise info 'f5:%',f5; openGauss\$\$ return 1; openGauss\$\$ end; openGauss\$\$ / CREATE FUNCTION openGauss=# 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>

Compatibility Configuration Item	Compatibility Behavior Control
	<pre>----- 1 (1 row)</pre> As shown above, f3 is filled with an incorrect default value. WARNING In this scenario, a non-default parameter is filled with the previous default value.
plsql_rollback_keep_user	Determines whether ROLLBACK and ROLLBACK TO SAVEPOINT in PL/SQL rolls back the current user. If this parameter is enabled, ROLLBACK in the PL/SQL does not change the current user. Example: <pre>openGauss=# CREATE USER plsql_rollback1 password 'huawei@123'; openGauss=# CREATE USER plsql_rollback2 password 'huawei@123'; openGauss=# GRANT plsql_rollback1 to plsql_rollback2; openGauss=# CREATE OR REPLACE procedure plsql_rollback1.p1 () authid definer openGauss=# as openGauss\$\$ va int; openGauss\$\$ begin openGauss\$\$ raise info 'current usr:%', current_user; openGauss\$\$ rollback; openGauss\$\$ raise info 'current usr:%', current_user; openGauss\$\$ end; openGauss\$\$ / CREATE PROCEDURE openGauss=# SET session AUTHORIZATION plsql_rollback2 PASSWORD 'huawei@123'; SET openGauss=> SET behavior_compat_options = 'plsql_rollback_keep_user'; SET openGauss=> CALL plsql_rollback1.p1 (); INFO: current usr:plsql_rollback1 INFO: current usr:plsql_rollback1 p1 ----- (1 row)</pre> CAUTION This parameter is valid only in the ORA compatibility mode.

Compatibility Configuration Item	Compatibility Behavior Control
dynamic_sql_check	After the parameter is enabled, the number of template parameters in the template SQL statement is compared with the number of variables in the USING clause. If they are inconsistent, an error is reported. <pre>openGauss=# create table tb_t1(col1 varchar2,col2 varchar2,col3 varchar2); CREATE TABLE gaussdb=# insert into tb_t1 select '1','1','2'; INSERT 0 1 openGauss=# create or replace procedure proc_test()as openGauss\$# v_a varchar2; openGauss\$# v_b varchar2; openGauss\$# v_cnt int; openGauss\$# v_sql varchar2; openGauss\$# begin openGauss\$# v_a = '1'; openGauss\$# v_b = '2'; openGauss\$# v_sql = 'select count(1) from tb_t1 where col1 = :va and col2 = :va and col3 = :vb;'; openGauss\$# execute immediate v_sql into v_cnt using v_a,v_a,v_b; openGauss\$# raise info 'v_cnt:%',v_cnt; openGauss\$# end; openGauss\$# / CREATE PROCEDURE openGauss=# call proc_test(); INFO: v_cnt:0 proc_test ----- (1 row) openGauss=# set behavior_compat_options='dynamic_sql_check'; SET openGauss=# call proc_test(); ERROR: argnum not match in Dynamic SQL, using args num : 3 , actual sql args num : 2 CONTEXT: SQL statement "select count(1) from tb_t1 where col1 = :va and col2 = :va and col3 = :vb;" PL/SQL function proc_test() line 10 at EXECUTE statemen</pre> CAUTION If the dynamic_sql_compat option is enabled, the dynamic_sql_check option does not take effect.

a_format_version

Parameter description: Specifies the database platform compatibility configuration item. The value of this parameter is an enumerated string.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: a string

Default value: ""

NOTE

Set a character string for the compatibility configuration item, for example, **set a_format_version='10c'**.

Table 18-12 Compatibility configuration items

Compatibility Configuration Item	Compatibility Behavior Control
10c	Compatible version of platform A

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: a string

Default value: ""

NOTE

Set a character string for the compatibility configuration item, for example, **set a_format_dev_version='s1'**.

Table 18-13 Compatibility configuration items

Compatibility Configuration Item	Compatibility Behavior Control
s1	After this parameter is enabled, the int casted from text can be rounded off.

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 instructions provided in [Table 18-2](#).

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

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: DD-Mon-YYYY HH:MI:SS.FF AM

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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.

18.3.16 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

Value range: an integer ranging from 8 to 128. The unit is KB.

Default value: 8 KB

max_cn_temp_file_size

Parameter description: Specifies the maximum number of temporary files that can be used by the CN during automatic SQL statement retries. The value 0 indicates that no temporary file is used.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 0 to 10485760. The unit is KB.

Default value: 5 GB

retry_ecode_list

Parameter description: Specifies the list of SQL error types that support automatic retries.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: a string

Default value: YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009 YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009 YY016

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

18.3.17 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 vacuuming mechanism is triggered to reduce the number of connections on each node to the value of **minimum_pool_size**.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 0 to 2147483647. The smallest unit is s.

Default value: 10 min (600 seconds)

minimum_pool_size

Parameter description: Specifies the minimum number of remaining connections in the pool on each node after the automatic vacuuming is triggered. If this parameter is set to **0**, the automatic vacuuming is disabled.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 needs 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)
- Finance edition (standard):
32000 (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, and 80-core CPU/640 GB memory); 16000 (72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 8000 (32-core CPU/256 GB memory); 4000 (16-core CPU/128 GB memory); 2000 (8-core CPU/64 GB memory)
- Enterprise edition:
CN: 5000 (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 4000 (96-core CPU/768 GB memory); 3000 (80-core CPU/640 GB memory); 2048 (80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB

memory, and 32-core CPU/256 GB memory); 1024 (16-core CPU/128 GB memory and 8-core CPU/64 GB memory)

DN: 20000 (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, and 96-core CPU/1024 GB memory); 15000 (96-core CPU/768 GB memory and 80-core CPU/640 GB memory); 10000 (80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); 5000 (32-core CPU/256 GB memory); 2000 (16-core CPU/128 GB memory); 1024 (8-core CPU/64 GB memory)

- Finance edition (data computing):

CN: 3000 (96-core CPU/768 GB memory); 2048 (72-core CPU/576 GB memory and 64-core CPU/512 GB memory); 1024 (32-core CPU/256 GB memory)

DN: 2048 (96-core CPU/768 GB memory, 72-core CPU/576 GB memory, and 64-core CPU/512 GB memory); 1024 (32-core CPU/256 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in [Table 18-2](#).

Setting suggestions: 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 is a BACKEND parameter. Set it based on instructions provided in [Table 18-2](#).

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: 2–1024

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 18-2](#).

Setting suggestions: 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 nodes 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 18-2](#).

Setting suggestions: Set this parameter based on the actual cluster specifications. If the value of this parameter is less than the number of DN nodes 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a BACKEND parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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

18.3.18 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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SUSERSET parameter. Set it based on method 3 in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: an integer ranging from 1 to 2147483647. The unit is s.

Default value: 1 min

enable_redistribute

Parameter description: Specifies whether unmatched nodes are redistributed.

This is a SUSET parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

This parameter is used for CM deployment. Do not set it.

Value range: 0 to 2

- **0** indicates the lab feature. For details, see *Lab Feature Description*.
- **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 DN's 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 18-2](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#). 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-2](#).

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

18.3.19 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-2](#).

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: Determines whether to enable primary/standby synchronization for logical replication slots.

This is a USERSET parameter. Set it based on instructions provided in [Table 18-2](#).

Value range: Boolean

- **on** indicates that primary/standby synchronization is enabled for logical replication slots.
- **off** indicates that primary/standby synchronization is disabled for logical 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 equal to or greater than twice of **max_changes_in_memory**.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on method 3 provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions in [Table 18-1](#).

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

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 is a SIGHUP parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.20 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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

NOTE

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 smaller than 16384.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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. 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.

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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: If this parameter is enabled, the system ignores the failure (but still generates an alarm). Continuing execution may result in breakdown, damaged data being transferred or saved, failure of data recovery from remote nodes, or other serious problems. You are not advised to modify the settings.

This is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-1](#).

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 18-1](#).

Value range: an integer ranging from 0 to 2147. 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.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 key is converted from a char-type data distribution into a varchar- or text-type data distribution.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 architecture.

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 backend. This parameter is used to prevent the Xlog file initialization from affecting the performance during transaction commission. However, such a fault may occur only when the system is overloaded. Therefore, you do not need to set this parameter.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 backend 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. The unit is KB.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 1 to 1024

Default value: 8

default_index_kind

Parameter description: Controls the default behavior of creating indexes.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer. Currently, only the fixed values **0**, **1**, and **2** are supported.

- **0**: The global partitioned 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.

18.3.21 Auditing

18.3.21.1 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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. A path relative to the **data** directory. Only the **sysadmin** user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 DNPs. 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: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

audit_data_format

Parameter description: Audits the format of log files. Currently, only the binary format is supported. Only the **sysadmin** user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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**. Ensure that the value of **audit_space_limit** is greater than the product of values of **audit_thread_num** and **audit_rotation_size**.
-

audit_file_remain_threshold

Parameter description: Specifies the maximum number of audit files in the audit directory.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 equal to or greater than 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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.

18.3.21.2 User and Permission Audit

audit_login_logout

Parameter description: Specifies whether to audit the GaussDB user's login (including login success and failure) and logout.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

NOTE

DNs do not audit logins and logouts from CNs.

audit_database_process

Parameter description: Specifies whether to audit the GaussDB user's login (including login success and failure) and logout.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer, **0** or **1**

- **0** indicates that the function of auditing GaussDB start, stop, recovery, and switchover operations of a database is disabled.
- **1** indicates that the function of auditing GaussDB start, stop, recovery, and switchover operations of a database is enabled.

Default value: 1

audit_user_locked

Parameter description: Specifies whether to audit the GaussDB user's locking and unlocking.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer, **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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer, **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 reclaiming of the GaussDB user's permission.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer, **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

18.3.21.3 Operation Auditing

audit_system_object

Parameter description: Specifies whether to audit the CREATE, DROP, and ALTER operations on the GaussDB database object. The GaussDB 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 134217727

- **0** indicates that the function of auditing the CREATE, DROP, and ALTER operations on the GaussDB database object is disabled.
- Other values indicate that the CREATE, DROP, and ALTER operations on a certain or some GaussDB database objects are audited.

Value description:

The value of this parameter is calculated by 27 binary bits. The 27 binary bits represent 27 types of GaussDB 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 27 binary bits, see [Table 18-14](#).

Default value: **67121159**, indicating that DDL operations on databases, schemas, users, data sources, and node groups are audited.

Table 18-14 audit_system_object parameter description

Binary Bit	Description	Value Range
Bit 0	Whether to audit the CREATE, DROP, and ALTER operations on databases.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 1	Whether to audit the CREATE, DROP, and ALTER operations on schemas.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 2	Whether to audit the CREATE, DROP, and ALTER operations on users and user mappings.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 3	Whether to audit the CREATE, DROP, ALTER, and TRUNCATE operations on tables.	• 0 indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are not audited. • 1 indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are audited.
Bit 4	Whether to audit the CREATE, DROP, and ALTER operations on indexes.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.

Binary Bit	Description	Value Range
Bit 5	Whether to audit the CREATE and DROP operations on views and materialized views.	• 0 indicates that the CREATE and DROP operations on these objects are not audited. • 1 indicates that the CREATE and DROP operations on these objects are audited.
Bit 6	Whether to audit the CREATE, DROP, and ALTER operations on triggers.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 7	Whether to audit the CREATE, DROP, and ALTER operations on procedures and functions.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 8	Whether to audit the CREATE, DROP, and ALTER operations on tablespaces.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 9	Whether to audit the CREATE, DROP, and ALTER operations on resource pools. For details, see section "Resource Load Management" in <i>Lab Feature Description</i> .	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 10	Whether to audit the CREATE, DROP, and ALTER operations on the WORKLOAD object. For details, see section "Resource Load Management" in the <i>Lab Feature Description</i> .	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 11	Reserved	-

Binary Bit	Description	Value Range
Bit 12	Whether to audit the CREATE, DROP, and ALTER operations on data sources. You are advised not to use it.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 13	Whether to audit the CREATE and DROP operations on node groups.	• 0 indicates that the CREATE and DROP operations on these objects are not audited. • 1 indicates that the CREATE and DROP operations on these objects are audited.
Bit 14	Whether to audit the CREATE, DROP, and ALTER operations on row-level security objects.	• 0 indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 15	Whether to audit the CREATE, DROP, and ALTER operations on types.	• 0 indicates that the CREATE, DROP, and ALTER operations on types are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on types are audited.
Bit 16	Whether to audit the CREATE, DROP, and ALTER operations on text search objects (CONFIGURATION and DICTIONARY).	• 0 indicates that the CREATE, DROP, and ALTER operations on text search objects are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on text search objects are audited.
Bit 17	Whether to audit the CREATE, DROP, and ALTER operations on directories.	• 0 indicates that the CREATE, DROP, and ALTER operations on directories are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on directories are audited.
Bit 18	Whether to audit the CREATE, DROP, and ALTER operations on synonyms.	• 0 indicates that the CREATE, DROP, and ALTER operations on synonyms are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on synonyms are audited.

Binary Bit	Description	Value Range
Bit 19	Whether to audit the CREATE, DROP, and ALTER operations on sequences.	• 0 indicates that the CREATE, DROP, and ALTER operations on sequences are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on sequences are audited.
Bit 20	Reserved	-
Bit 21	Whether to audit the CREATE, DROP, and ALTER operations on package objects. (Currently, the distributed system does not support package objects.)	• 0 indicates that the CREATE, DROP, and ALTER operations on packages are not audited. • 1 indicates that the CREATE, DROP, and ALTER operations on packages are audited.
Bit 22	Reserved	-
Bit 23	Reserved	-
Bit 24	Whether to audit the ALTER and DROP operations on the gs_global_config objects.	• 0 indicates that the ALTER and DROP operations on the gs_global_config objects are not audited. • 1 indicates that the ALTER and DROP operations on the gs_global_config objects are audited.
Bit 25	Reserved	-
Bit 26	Reserved	-

audit_dml_state

Parameter description: Specifies whether to audit the INSERT, UPDATE, and DELETE operations on a specific table.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer, 0 or 1

- 0 indicates that auditing the stored procedure or function execution is disabled.
- 1 indicates that auditing the stored procedure or function execution is enabled.

Default value: 0

audit_copy_exec

Parameter description: Specifies whether to audit the COPY operation.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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.

18.3.22 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483. The unit is s.

Default value: 10 min

 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.

18.3.23 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 18-2](#).

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.

18.3.23.1 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 this parameter, see [Table 18-2](#).

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 18-2](#).

Default value: 16 MB

log_min_messages

Parameter description: Specifies which message levels are written to the CM Agent 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 18-2](#).

Default value: warning

incremental_build

Parameter description: Specifies whether a standby DN is incrementally built. If this parameter is enabled, a standby DN is incrementally built.

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 18-2](#).

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 18-2](#).

- 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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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 second. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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 For details about how to modify this parameter, see [Table 18-2](#).

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 terminates all instances on the node.

Value range: an integer For details about how to modify this parameter, see [Table 18-2](#).

Default value: 0, indicating that the operation of terminating 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. For details about how to modify this parameter, see [Table 18-2](#).

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**. For details about how to modify this parameter, see [Table 18-2](#).

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 For details about how to modify this parameter, see [Table 18-2](#).

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 . Any modification of this parameter takes effect only after the cluster and CM Agent are restarted. Only Arm is supported. For details about how to modify this parameter, see [Table 18-2](#).

Value range: an integer ranging from 0 to 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 this parameter, see [Table 18-2](#).

- **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.

Value range: an integer ranging from 0 to $2^{31} - 1$. The unit is second. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

Default value: 90

enable_log_compress

Parameter description: Specifies whether to enable log compression.

Value range: Boolean For details about how to modify this parameter, see [Table 18-2](#).

- **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 For details about how to modify this parameter, see [Table 18-2](#).

- **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 this parameter, see [Table 18-2](#).

- **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: an integer, **0** or **1**. The value **1** indicates enabled, while the value **0** indicates disabled. For details about how to modify this parameter, see [Table 18-2](#).

Default value:

Independent deployment: **1**

Finance edition (standard), enterprise edition, and finance edition (data computing): **0**

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify this parameter, see [Table 18-2](#).

- **0**: disabled.
- **1**: enabled.

Default value: off

unix_socket_directory

Parameter description: Specifies the directory location of the Unix socket.

Value range: a string. For details about how to modify this parameter, see [Table 18-2](#).

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 For details about how to modify this parameter, see [Table 18-2](#).

- **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, 0). The value range for each number is [0,100]. The unit is %. Value **0** indicates that the detection is disabled. For details about how to modify this parameter, see [Table 18-2](#).

Default value: (0,0,0,0,0)

18.3.23.2 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 the CM Server is restarted. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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**. For details about how to modify this parameter, see [Table 18-2](#).

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 the CM Server is restarted. For details about how to modify this parameter, see [Table 18-2](#).

Default value: 1000

alarm_component

Parameter description: Specifies the location of the alarm component that processes alarms.

Value range: a string For details about how to modify this parameter, see [Table 18-2](#).

- 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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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**. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. 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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-1](#).

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**. For details about how to modify this parameter, see [Table 18-1](#).

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 character string. For details about how to modify the parameter, see [Table 18-1](#).

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. For details about how to modify this parameter, see [Table 18-1](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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: an integer, **0** or **1**. For details about how to modify this parameter, see [Table 18-2](#).

Default value: 0

switch_rto

Parameter description: Specifies the delay for the forcible startup 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 s. The minimum value that takes effect is 60. If this parameter is set to a value less than 60, 60s is used. For details about how to modify this parameter, see [Table 18-2](#).

Default value: 600

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 the CM Server is restarted. This parameter cannot be enabled for non-DR clusters. For details about how to modify this parameter, see [Table 18-2](#).

- **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 ≤ 10 s; RTO for combined faults ≤ 30 s).

Value range: an integer, **0** or **1**. The value **1** indicates enabled, while the value **0** indicates disabled. For details about how to modify this parameter, see [Table 18-2](#).

Default value:

Independent deployment: **1**

Finance edition (standard), enterprise edition, and finance edition (data computing): **0**

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. For details about how to modify this parameter, see [Table 18-2](#).

Default value: 180

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean Any modification of this parameter takes effect only after the CM Server is restarted. For details about how to modify this parameter, see [Table 18-2](#).

- 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 the CM Server is restarted. For details about how to modify this parameter, see [Table 18-2](#).

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 a restart. For details about how to modify this parameter, see [Table 18-2](#).

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 a restart. For details about how to modify this parameter, see [Table 18-2](#).

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 a restart. For details about how to modify this parameter, see [Table 18-2](#).

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**. For details about how to modify this parameter, see [Table 18-2](#).

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, in the range [1,100]. For details about how to modify this parameter, see [Table 18-2](#).

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]. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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. For details about how to modify this parameter, see [Table 18-2](#).

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). For details about how to modify this parameter, see [Table 18-2](#).

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 the CM Server is restarted. This parameter cannot be enabled for non-DR clusters. For details about how to modify this parameter, see [Table 18-1](#).

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.

18.3.24 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a character string, which complies with the identifier naming rules (For details, see "Identifier Naming Rules" in "SQL Reference" > "Keywords" in *Developer Guide*).

Default value: NULL

port

Parameter description: Specifies the host port number listened by the primary or standby GTM.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a character string, which complies with the identifier naming rules (For details, see "Identifier Naming Rules" in "SQL Reference" > "Keywords" in *Developer Guide*).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a character string, which complies with the identifier naming rules (For details, see "Identifier Naming Rules" in "SQL Reference" > "Keywords" in *Developer Guide*).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a character string, which complies with the identifier naming rules (For details, see "Identifier Naming Rules" in "SQL Reference" > "Keywords" in *Developer Guide*).

Default value: NULL

active_port

Parameter description: Specifies the port number of the target GTM server.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 5 to 2147483647. The unit is s.

Default value: 5

keepalives_count

Parameter description: Specifies the number of keepalive 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647

Default value: 0, indicating that the connection is immediately broken if no keepalive signal from the client is received by the GTM.

keepalives_idle

Parameter description: Specifies the interval for sending keepalive signals.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 512 to 2147483647. The unit is 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 18-1](#).

Default value: gtm.conf

data_dir

Parameter description: Specifies the GTM data file directory.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 about the parameters, see [Table 18-8](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a non-negative integer. The unit is second.

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 18-1](#).

- 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 18-1](#).

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 18-1](#).

- **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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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}/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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: "

gtm_option

Parameter description: Specifies the GTM mode. This parameter must be configured on all GTMs, CNs, and DNPs. The value must be the same. There are three GTM modes: GTM, GTM-lite, and GTM-free. 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 18-1](#).

Default value: 1

csn_sync_interval

Parameter description: Specifies the interval for synchronizing CSN between the primary and standby GTMs, in unit of s.

Value range: an integer ranging from 1 to 2147483647. Set it based on instructions provided in [Table 18-1](#).

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 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 16384

Default value: 1024

18.3.25 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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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.

18.3.26 Miscellaneous Parameters

server_version

Parameter description: Specifies the server version number.

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.

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 is a SUSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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: Specifies the deployment region of the entire cluster. It must contain only the characters specified in RFC3986, and the maximum length is 2047 bytes. This parameter applies only to the DWS scenario in the current version.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: 0-1024

Default value: 10

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- The value **on** indicates that the global index is updated regardless of whether the partition DDL commands contain the UPDATE GLOBAL INDEX clause.
- The value **off** indicates that the global index is not updated unless the partition DDL commands contain the UPDATE GLOBAL INDEX clause.

Default value: off

18.3.27 Wait Event

enable_instr_track_wait

Parameter description: Specifies whether to enable real-time collection of wait event information.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.28 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 is greater than the value of **instr_unique_sql_count**, the extra unique SQL records are not collected.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is an INTERNAL parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: enumerated values

top: Only top-level SQL statements are recorded.

Default value: top

unique_sql_retention_time

Parameter description: Specifies the memory cleanup interval for the unique SQL hash table. The default value is 30 minutes.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 1 to 3650

Default value: 30 min

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is an INTERNAL parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

enable_slow_query_log (Discarded)

Parameter description: Specifies whether to write the slow query information to the log file. This parameter is discarded in this version.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on:** indicates that slow query information needs to be written into log files.
- **off:** indicates that slow query information does not need to be written into log files.

Default value: on

query_log_file (Discarded)

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 function has been discarded in this version.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 (Discarded)

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 discarded in this version.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

NOTICE

If this parameter is set to an invalid path, the cluster cannot be started.

 NOTE

Valid path: Users have read and write permissions on the path.

Invalid path: Users 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

NOTICE

If this parameter is set to an invalid path, the cluster cannot be started.

 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

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

 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

enable_stmt_track

Parameter description: Specifies whether to enable the full/slow SQL statement feature.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 18-2](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 0 to 2147483647

Default value: 1000

track_stmt_details_size

Parameter description: Specifies the maximum size (in bytes) of execution events that can be collected by a single statement.

This is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 0 to 100000000

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

This parameter consists 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.
- **slow sql retention time** indicates the retention period of slow SQL statements. The value ranges from 0 to 604800.

Default value: 3600,604800

track_stmt_stat_level

Parameter description: Controls the level of statement execution tracing.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 18-1](#). 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 the higher level (L2 > L1 > L0) of the full SQL statements and slow SQL statements.

Default value: OFF,L0

18.3.29 System Performance Snapshot

enable_wdr_snapshot

Parameter description: Specifies whether to enable the database monitoring snapshot function.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 1 to 8. 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 a result is not returned within the specified time, the snapshot operation fails.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: an integer ranging from 100 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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: an integer ranging from 10000 to 100000

Default value: 100000

asp_sample_interval

Parameter description: Specifies the sampling interval.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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**, it indicates that the update ratio is 10:1.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: a string

Default value: asp-%Y-%m-%d_%H%M%S.log

18.3.30 Security Configuration

enable_security_policy

Parameter description: Specifies whether the unified audit and dynamic data masking policies take effect.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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.

This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Value range: a string

Default value: 'https:127.0.0.1'

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

18.3.31 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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Value range: Boolean

- **on** or **true** indicates that memory optimization is enabled.
- **off** or **false** indicates that memory optimization is disabled.

Default value: off

18.3.32 User-defined Functions

udf_memory_limit

Parameter description: Specifies the maximum physical memory that can be used when CNs and DN execute 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a USERSET parameter. Set it based on instructions in [Table 18-1](#).

Setting suggestions: Contact an administrator.

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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

Setting suggestions: Contact an administrator.

Value range: an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB.

Default value: 1 GB

18.3.33 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 is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

Value range: Boolean

- **on** indicates that the job thread is not started.
- **off** indicates that the job thread is started.

Default value: off

18.3.34 Thread Pool

enable_thread_pool

Parameter description: Specifies whether to enable the thread pool function. This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 the sysadmin user can access this parameter. This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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** is the total number of initial threads in the thread pool, which can be dynamically expanded. 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 x 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)
- Finance edition (standard):
CN: '768,2,(nobind)' (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); '684,2,(nobind)' (96-core CPU/1024 GB memory and 96-core CPU/768 GB memory); '512,2,(nobind)' (80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); '256,2,(nobind)' (32-core CPU/256 GB memory); '128,2,(nobind)' (16-core CPU/128 GB memory); '64,2,(nobind)' (8-core CPU/64 GB memory)
DN: '4096,2,(nobind)' (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); '2048,2,(nobind)' (32-core CPU/256 GB memory); '1024,2,(nobind)' (16-core CPU/128 GB memory); '512,2,(nobind)' (8-core CPU/64 GB memory)
- Enterprise edition:
CN: '768,2,(nobind)' (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); '512,2,(nobind)' (96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); '256,2,(nobind)' (32-core CPU/256 GB memory); '128,2,(nobind)' (16-core CPU/128 GB memory); '64,2,(nobind)' (8-core CPU/64 GB memory)
DN: '1536,2,(nobind)' (128-core CPU/1024 GB memory and 104-core CPU/1024 GB memory); '1024,2,(nobind)' (96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 80-core CPU/512 GB memory, 72-core CPU/576 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); '512,2,(nobind)' (32-core CPU/256 GB memory); '256,2,(nobind)' (16-core CPU/128 GB memory); '128,2,(nobind)' (8-core CPU/64 GB memory)
- Finance edition (data computing):
CN: 2 GB (96-core CPU/768 GB memory); '256,2,(nobind)' (72-core CPU/576 GB memory); '128,2,(nobind)' (64-core CPU/512 GB memory); '64,2,(nobind)' (32-core CPU/256 GB memory)
DN: '2048,2,(nobind)' (96-core CPU/768 GB memory); '1024,2,(nobind)' (72-core CPU/576 GB memory); '512,2,(nobind)' (64-core CPU/512 GB memory); '256,2,(nobind)' (32-core CPU/256 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 the **sysadmin** user can access this parameter. This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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: Value of **stream_proc_ratio** x 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 accumulated sessions in the thread pool for escape from overload. This parameter takes effect only when the GUC parameters **enable_thread_pool** and **use_workload_manager** are enabled. This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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**: Percentage of the number of sessions that are recovered to the normal state in the initial number of threads in the thread pool. When the number of accessed sessions is less than the initial number of threads in the thread pool multiplied by 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**: Percentage of the number of accessed sessions to the initial number of threads in the thread pool when the thread pool is overloaded. If the number of accessed sessions is greater than the initial number of threads in the thread pool multiplied by 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 to access the thread pool. 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 = '100,200'
```

When the number of stacked sessions exceeds 200% of the initial number of threads in the thread pool, new connections are forbidden and stacked sessions are killed. When the number of stacked sessions is less than 100% of the initial number of threads in the thread pool, new connections are allowed.

NOTICE

- The number of stacked sessions can be obtained by querying the number of data records in the **pg_stat_activity** view. A few backend threads need to be filtered out. 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.
-

18.3.35 Backup and Restoration

operation_mode

Parameter description: Specifies whether the system enters the backup and restoration mode.

This is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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

18.3.36 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 18-2](#).

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

18.3.37 Global SysCache Parameters

enable_global_syscache

Parameter description: Specifies whether to enable the global system cache function. This is a POSTMASTER parameter. Set it based on instructions in [Table 18-1](#).

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 is a SIGHUP parameter. Set it based on instructions provided in [Table 18-1](#).

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.

18.3.38 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_space_limit_size
undo_limit_size_per_transaction
undo_zone_count
ustore_attr
enable_ustore
walwriter_cpu_bind

Obsolete Functions

enable_adio_debug
enable_adio_function
enable_fast_allocate
prefetch_quantity
backwrite_quantity
cstore_prefetch_quantity

cstore_backwrite_quantity
cstore_backwrite_max_threshold
fast_extend_file_size
effective_io_concurrency