

Data Warehouse Service Database

Developer Guide

Issue 21
Date 2020-12-30



Copyright © Huawei Technologies Co., Ltd. 2020. 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 Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks 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 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.

Contents

1 Welcome.....	1
1.1 Target Readers.....	1
1.2 Reading Guide.....	2
1.3 Conventions.....	3
1.4 Prerequisites.....	4
2 System Overview.....	5
2.1 Product Architecture.....	5
2.2 HA Transaction Processing.....	8
2.3 High Query Performance.....	9
2.4 Technical Specifications.....	10
2.5 SQL Syntax.....	10
2.6 Related Concepts.....	11
3 Syntax Compatibility Differences Between Teradata and Oracle.....	13
4 Database Security Management.....	15
4.1 Managing Users and Their Permissions.....	15
4.1.1 Default Permission Mechanism.....	15
4.1.2 System Administrator.....	16
4.1.3 Separation of Duty.....	16
4.1.4 Users.....	18
4.1.5 Roles.....	19
4.1.6 Schema.....	20
4.1.7 User Permission Setting.....	21
4.1.8 Row-Level Access Control.....	22
4.1.9 Data Redaction.....	23
4.2 Setting Account Security Policies.....	25
4.2.1 Setting Account Security Locking Policies.....	25
4.2.2 Setting the Validity Period of an Account.....	26
4.2.3 Setting Password Security Policy.....	27
4.3 Viewing Audit Information.....	29
5 Development and Design Proposal.....	31
5.1 Development and Design Proposal.....	31
5.2 Database Object Naming Conventions.....	31

5.3 Database Object Design.....	32
5.3.1 Database and Schema Design.....	32
5.3.2 Table Design.....	33
5.3.3 Column Design.....	35
5.3.4 Constraint Design.....	37
5.3.5 View and Joined Table Design.....	38
5.4 JDBC Configuration.....	38
5.5 SQL Compilation.....	39
6 Tutorial: Development Using JDBC or ODBC.....	44
6.1 Development Specifications.....	44
6.2 Downloading Drivers.....	44
6.3 Using JDBC for Development.....	44
6.3.1 JDBC Development Process.....	44
6.3.2 Example: Common Operations.....	44
6.3.3 Example: Retrying SQL Queries for Applications.....	48
6.3.4 Example: Importing and Exporting Data Through Local Files.....	51
6.3.5 Example: Migrating Data from MySQL to GaussDB(DWS).....	52
6.4 JDBC Interface Reference.....	54
6.4.1 java.sql.Connection.....	54
6.4.2 java.sql.CallableStatement.....	55
6.4.3 java.sql.DatabaseMetaData.....	56
6.4.4 java.sql.Driver.....	58
6.4.5 java.sql.PreparedStatement.....	59
6.4.6 java.sql.ResultSet.....	60
6.4.7 java.sql.ResultSetMetaData.....	62
6.4.8 java.sql.Statement.....	62
6.4.9 javax.sql.ConnectionPoolDataSource.....	63
6.4.10 javax.sql.DataSource.....	64
6.4.11 javax.sql.PooledConnection.....	64
6.4.12 javax.naming.Context.....	65
6.4.13 javax.naming.spi.InitialContextFactory.....	65
6.4.14 CopyManager.....	66
6.5 Using ODBC for Development.....	67
6.5.1 ODBC Development Process.....	67
6.5.2 Examples.....	67
6.6 ODBC Interfaces	72
6.6.1 SQLAllocEnv.....	73
6.6.2 SQLAllocConnect.....	73
6.6.3 SQLAllocHandle.....	73
6.6.4 SQLAllocStmt.....	74
6.6.5 SQLBindCol.....	74
6.6.6 SQLBindParameter.....	76

6.6.7 SQLColAttribute.....	77
6.6.8 SQLConnect.....	78
6.6.9 SQLDisconnect.....	80
6.6.10 SQLExecDirect.....	80
6.6.11 SQLExecute.....	81
6.6.12 SQLFetch.....	82
6.6.13 SQLFreeStmt.....	83
6.6.14 SQLFreeConnect.....	83
6.6.15 SQLFreeHandle.....	83
6.6.16 SQLFreeEnv.....	84
6.6.17 SQLPrepare.....	84
6.6.18 SQLGetData.....	85
6.6.19 SQLGetDiagRec.....	87
6.6.20 SQLSetConnectAttr.....	89
6.6.21 SQLSetEnvAttr.....	90
6.6.22 SQLSetStmtAttr.....	91
7 Data Import.....	93
7.1 Import Modes.....	93
7.2 Concurrently Importing Data from OBS.....	96
7.2.1 About Parallel Data Import from OBS.....	96
7.2.2 Creating Access Keys (AK and SK).....	101
7.2.3 Uploading Data to OBS.....	103
7.2.4 Creating an OBS Foreign Table.....	104
7.2.5 Importing Data.....	107
7.2.6 Handling Import Errors.....	108
7.2.7 Example.....	111
7.3 Using GDS to Import Data from a Remote Server.....	113
7.3.1 Importing Data In Parallel Using GDS.....	113
7.3.2 Preparing Source Data.....	117
7.3.3 Installing, Configuring, and Starting GDS.....	118
7.3.4 Creating a GDS Foreign Table.....	122
7.3.5 Importing Data.....	125
7.3.6 Handling Import Errors.....	126
7.3.7 Stopping GDS.....	129
7.3.8 Example.....	130
7.4 Running the INSERT Statement to Insert Data.....	133
7.5 Running the COPY FROM STDIN Statement to Import Data.....	134
7.5.1 Data Import Using COPY FROM STDIN.....	134
7.5.2 Introduction to the CopyManager Class.....	134
7.5.3 Example 1: Importing and Exporting Data Through Local Files.....	135
7.5.4 Example 2: Migrating Data from a MySQL Database to GaussDB(DWS).....	137
7.6 Using a gsql Meta-Command to Import Data.....	139

7.7 Importing Data From MRS to a Data Warehouse Cluster.....	142
7.7.1 Overview of Importing Data from MRS.....	142
7.7.2 Preparing Data in an MRS Cluster.....	142
7.7.3 Manually Creating a Foreign Server.....	145
7.7.4 Creating a Foreign Table.....	149
7.7.5 Importing Data.....	154
7.7.6 Deleting Resources.....	156
7.7.7 Error Handling.....	157
7.8 Importing Stream Data from DIS to GaussDB(DWS).....	157
7.9 Using CDM to Migrate Data to GaussDB(DWS).....	159
7.10 Using the DSC Tool to Migrate SQL Scripts.....	160
7.11 Checking for Data Skew.....	161
7.12 Analyzing a Table.....	163
7.13 Doing VACUUM to a Table.....	164
7.14 Managing Concurrent Write Operations.....	165
7.14.1 Transaction Isolation.....	165
7.14.2 Write and Read/Write Operations.....	165
7.14.3 Potential Deadlocks During Concurrent Write.....	166
7.14.4 Concurrent INSERT and DELETE in the Same Table.....	166
7.14.5 Concurrent INSERT in the Same table.....	167
7.14.6 Concurrent UPDATE in the Same Table.....	167
7.14.7 Concurrent Data Import and Queries.....	168
8 Data Export.....	169
8.1 Exporting Data to OBS in Parallel.....	169
8.1.1 Parallel OBS Data Export.....	169
8.1.2 Planning Data Export.....	173
8.1.3 Creating an OBS Foreign Table.....	175
8.1.4 Exporting Data.....	176
8.1.5 Examples.....	177
8.2 Using GDS to Export Data to a Remote Server.....	180
8.2.1 Exporting Data In Parallel Using GDS.....	180
8.2.2 Planning Data Export.....	183
8.2.3 Installing, Configuring, and Starting GDS.....	184
8.2.4 Creating a GDS Foreign Table.....	184
8.2.5 Exporting Data.....	185
8.2.6 Stopping GDS.....	186
8.2.7 Example.....	186
8.3 Using gs_dump and gs_dumpall to Export Data.....	189
8.3.1 Overview.....	189
8.3.2 Exporting a Single Database.....	191
8.3.2.1 Exporting a Database.....	191
8.3.2.2 Exporting a Schema.....	194

8.3.2.3 Exporting a Table.....	196
8.3.3 Exporting All Databases.....	199
8.3.3.1 Exporting All Databases.....	199
8.3.3.2 Exporting Global Objects.....	201
8.3.4 Data Export By a User Without Required Permissions.....	202
9 Querying External Data.....	206
9.1 Querying Data on OBS.....	206
9.1.1 OBS Query Overview.....	206
9.1.2 Preparing Data on OBS.....	206
9.1.3 Creating a Foreign Server.....	208
9.1.4 Creating a Foreign Table.....	211
9.1.5 Querying Data on OBS Through Foreign Tables.....	213
9.1.6 Deleting Resources.....	214
9.2 Querying Data on MRS.....	216
10 Using Express to Query Data on OBS.....	217
10.1 Express Functions.....	217
10.2 Using Express Clusters.....	218
10.2.1 Example.....	218
10.2.2 Creating Foreign Servers and OBS Foreign Tables.....	218
10.2.3 Querying Data.....	219
10.2.4 Checking Whether a Query Is Accelerated.....	220
11 PostGIS Extension.....	221
11.1 PostGIS.....	221
11.2 Using PostGIS.....	221
11.3 PostGIS Support and Constraints.....	222
11.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS).....	226
12 Real-Time GaussDB(DWS).....	274
12.1 Time Series Engine.....	274
12.1.1 Overview.....	274
12.1.2 Usage.....	274
12.1.2.1 Partition Management.....	274
12.1.2.2 Time Series Function.....	276
12.1.3 Support and Constraints.....	283
13 Setting GUC Parameters.....	285
13.1 Viewing Parameter Values.....	285
13.2 Resetting Parameters.....	286
13.3 GUC Parameter Usage.....	287
13.4 Connection and Authentication.....	288
13.4.1 Connection Settings.....	288
13.4.2 Security and Authentication (postgresql.conf).....	289

13.4.3 Communication Library Parameters.....	289
13.5 Resource Consumption.....	291
13.5.1 Disk Space.....	291
13.5.2 Cost-based Vacuum Delay.....	292
13.5.3 Asynchronous I/O Operations.....	294
13.6 Parallel Data Import.....	296
13.7 Write Ahead Logs.....	297
13.7.1 Settings.....	297
13.8 HA Replication.....	298
13.8.1 Primary Server.....	298
13.9 Query Planning.....	299
13.9.1 Optimizer Method Configuration.....	299
13.9.2 Optimizer Cost Constants.....	308
13.9.3 Genetic Query Optimizer.....	310
13.9.4 Other Optimizer Options.....	312
13.10 Error Reporting and Logging.....	320
13.10.1 Logging Time.....	321
13.10.2 Logging Content.....	324
13.11 Statistics During the Database Running.....	328
13.11.1 Query and Index Statistics Collector.....	328
13.11.2 Performance Statistics.....	331
13.12 Workload Management.....	332
13.13 Default Settings of Client Connection.....	337
13.13.1 Statement Behavior.....	337
13.13.2 Zone and Formatting.....	342
13.13.3 Other Default Parameters.....	346
13.14 Lock Management.....	347
13.15 Version and Platform Compatibility.....	349
13.15.1 Compatibility with Earlier Versions.....	349
13.15.2 Platform and Client Compatibility.....	352
13.16 Fault Tolerance.....	353
13.17 Cluster Transaction Parameters.....	354
13.18 Developer Operations.....	357
13.19 Transaction Monitoring.....	368
13.20 Miscellaneous Parameters.....	369
14 Resource Load Management.....	377
14.1 Resource Load Management Overview.....	377
14.2 Memory Management.....	378
14.2.1 Memory Management Overview.....	378
14.2.2 Job-level Memory Control.....	379
14.3 Resource Load Management Framework Overview.....	380
14.3.1 Resource Pools.....	380

14.3.2 Association Tasks.....	387
14.4 Priority Scheduling.....	388
14.4.1 CPU Priority Scheduling.....	388
14.4.2 I/O Priority Scheduling.....	390
14.5 Storage Space Management.....	392
14.6 Resource Monitoring.....	392
14.6.1 Querying User Resources.....	393
14.6.2 Monitoring Memory Resources.....	394
14.6.3 Instance Resource Monitoring.....	396
14.6.4 Real-Time TopSQL.....	398
14.6.5 Historical TopSQL.....	400
14.6.6 Example for Querying for Top SQLs.....	404
15 PL/Java User-defined Functions.....	408
16 Query Performance Optimization.....	420
16.1 Query Performance Tuning Overview.....	420
16.2 Analyzing Queries.....	420
16.2.1 Query Execution Process.....	420
16.2.2 SQL Execution Plan.....	423
16.2.3 SQL Execution Plan Details.....	424
16.2.4 Querying SQL Statements That Affect Performance Most.....	431
16.2.5 Checking Blocked Statements.....	432
16.3 Tuning Queries.....	433
16.3.1 Tuning Process.....	433
16.3.2 Updating Statistics.....	434
16.3.3 Reviewing and Modifying a Table Definition.....	435
16.3.4 Typical SQL Optimization Methods.....	436
16.3.4.1 Typical SQL Optimization Methods.....	436
16.3.4.2 Optimizing SQL Self-Diagnosis.....	436
16.3.4.3 Optimizing Statement Pushdown.....	439
16.3.4.4 Optimizing Subqueries.....	446
16.3.4.5 Optimizing Statistics.....	447
16.3.4.6 Optimizing Operators.....	452
16.3.4.7 Optimizing Data Skew.....	453
16.3.5 Adjusting Key Parameters During SQL Tuning.....	459
16.3.6 Hint-based Tuning.....	461
16.3.6.1 Plan Hint Optimization.....	461
16.3.6.2 Join Order Hints.....	463
16.3.6.3 Join Operation Hints.....	464
16.3.6.4 Rows Hints.....	465
16.3.6.5 Stream Operation Hints.....	467
16.3.6.6 Scan Operation Hints.....	467
16.3.6.7 Sublink Name Hints.....	468

16.3.6.8 Skew Hints.....	469
16.3.6.9 Hint Errors, Conflicts, and Other Warnings.....	474
16.3.6.10 Plan Hint Cases.....	476
16.3.7 Routinely Maintaining Tables.....	480
16.3.8 Routinely Recreating an Index.....	481
16.3.9 Configuration SMP.....	482
16.3.9.1 SMP Application Scenarios and Restrictions.....	482
16.3.9.2 Resource Impact on SMP Performance.....	484
16.3.9.3 Other Factors Affecting SMP Performance.....	484
16.3.9.4 Recommended Suggestions for SMP.....	485
16.4 Optimization Cases.....	485
16.4.1 Case: Selecting an Appropriate Distribution Column.....	485
16.4.2 Case: Creating an Appropriate Index.....	486
16.4.3 Case: Adding NOT NULL for JOIN Columns.....	487
16.4.4 Case: Pushing Down Sort Operations to DNs.....	489
16.4.5 Setting cost_param and Optimizing Query Performance.....	490
16.4.6 Case: Adjusting the Distribution Key.....	494
16.4.7 Case: Adjusting the Partial Clustering Key.....	494
16.4.8 Case: Adjusting the Table Storage Mode in a Medium Table.....	495
16.4.9 Case: Adjusting the Local Clustering Column.....	496
16.4.10 Case: Reconstructing Partition Tables.....	497
16.4.11 Case: Adjusting the GUC Parameter best_agg_plan.....	498
16.4.12 Case: Rewriting SQL and Deleting Subqueries (Case 1).....	499
16.4.13 Case: Rewriting SQL and Deleting Subqueries (Case 2).....	500
16.4.14 Case: Rewriting SQL Statements and Eliminating Prune Interference.....	501
16.4.15 Case: Rewriting SQL Statements and Deleting in-clause.....	502
16.4.16 Case: Setting Partial Cluster Keys.....	503
16.5 Troubleshooting Errors Using SQL Statements.....	506
16.5.1 Low Query Efficiency.....	506
16.5.2 "failed to find conversion function from unknown to text" Is Displayed When a User Executes the SELECT Statement.....	507
16.5.3 DROP TABLE Fails to Be Executed.....	507
16.5.4 Different Data Is Displayed for the Same Table Queried By Multiple Users.....	508
16.5.5 An Error Occurs During the Integer Conversion.....	508
16.5.6 Automatic Retry upon SQL Statement Execution Errors.....	509
17 Stored Procedures.....	513
17.1 Data Types.....	513
17.2 Data Type Conversion.....	513
17.3 Arrays.....	514
17.4 record.....	515
17.5 Syntax.....	517
17.5.1 Basic Structure.....	518

17.5.2 Anonymous Block.....	518
17.5.3 Subprogram.....	519
17.6 Basic Statements.....	520
17.6.1 Variable Definition Statement.....	520
17.6.2 Assignment Statement.....	521
17.6.3 Call Statement.....	522
17.6.4 Return Statement.....	523
17.7 Dynamic Statements.....	524
17.7.1 Executing Dynamic Query Statements.....	525
17.7.2 Executing Dynamic Non-query Statements.....	527
17.7.3 Dynamically Calling Stored Procedures.....	528
17.7.4 Dynamically Calling Anonymous Blocks.....	530
17.8 Control Statements.....	531
17.8.1 Conditional Statements.....	531
17.8.2 Loop Statements.....	533
17.8.3 Branch Statements.....	537
17.8.4 NULL Statements.....	538
17.8.5 Error Trapping Statements.....	538
17.8.6 GOTO Statements.....	540
17.9 Lock Operations.....	542
17.10 Cursors.....	542
17.10.1 Overview.....	542
17.10.2 Explicit Cursor.....	543
17.10.3 Implicit Cursor.....	547
17.10.4 Cursor Loop.....	548
17.11 Advanced Packages.....	549
17.11.1 DBMS_LOB.....	549
17.11.2 DBMS_RANDOM.....	558
17.11.3 DBMS_OUTPUT.....	559
17.11.4 UTL_RAW.....	561
17.11.5 DBMS_JOB.....	563
17.11.6 DBMS_SQL.....	570
17.12 Debugging.....	581
18 System Catalogs and System Views.....	584
18.1 Overview of System Catalogs and System Views.....	584
18.2 System Catalogs.....	584
18.2.1 GS_OBSSCANINFO.....	584
18.2.2 GS_WLM_INSTANCE_HISTORY.....	585
18.2.3 GS_WLM_OPERATOR_INFO.....	586
18.2.4 GS_WLM_SESSION_INFO.....	588
18.2.5 GS_WLM_USER_RESOURCE_HISTORY.....	588
18.2.6 PG_AGGREGATE.....	589

18.2.7 PG_AM.....	590
18.2.8 PG_AMOP.....	592
18.2.9 PG_AMPROC.....	593
18.2.10 PG_APP_WORKLOADGROUP_MAPPING.....	594
18.2.11 PG_ATTRDEF.....	594
18.2.12 PG_ATTRIBUTE.....	595
18.2.13 PG_AUTHID.....	597
18.2.14 PG_AUTH_HISTORY.....	598
18.2.15 PG_AUTH_MEMBERS.....	599
18.2.16 PG_CAST.....	599
18.2.17 PG_CLASS.....	600
18.2.18 PG_COLLATION.....	604
18.2.19 PG_CONSTRAINT.....	605
18.2.20 PG_CONVERSION.....	607
18.2.21 PG_DATABASE.....	607
18.2.22 PG_DB_ROLE_SETTING.....	608
18.2.23 PG_DEFAULT_ACL.....	609
18.2.24 PG_DEPEND.....	609
18.2.25 PG_DESCRIPTION.....	611
18.2.26 PG_DIRECTORY.....	611
18.2.27 PG_ENUM.....	612
18.2.28 PG_EXTENSION.....	612
18.2.29 PG_EXTENSION_DATA_SOURCE.....	613
18.2.30 PG_FOREIGN_DATA_WRAPPER.....	613
18.2.31 PG_FOREIGN_SERVER.....	614
18.2.32 PG_FOREIGN_TABLE.....	615
18.2.33 PG_INDEX.....	615
18.2.34 PG_INHERITS.....	617
18.2.35 PG_JOB.....	617
18.2.36 PG_JOB_PROC.....	619
18.2.37 PG_LANGUAGE.....	619
18.2.38 PG_LARGEOBJECT.....	620
18.2.39 PG_LARGEOBJECT_METADATA.....	621
18.2.40 PG_NAMESPACE.....	621
18.2.41 PG_OBJECT.....	622
18.2.42 PG_OBSSCANINFO.....	623
18.2.43 PG_OPCLASS.....	623
18.2.44 PG_OPERATOR.....	624
18.2.45 PG_OPFAMILY.....	625
18.2.46 PG_PARTITION.....	626
18.2.47 PG_PLTEMPLATE.....	627
18.2.48 PG_PROC.....	628

18.2.49 PG_RANGE.....	631
18.2.50 PG_REDACTION_COLUMN.....	632
18.2.51 PG_REDACTION_POLICY.....	632
18.2.52 PG_RESOURCE_POOL.....	633
18.2.53 PG_REWRITE.....	634
18.2.54 PG_SECLABEL.....	635
18.2.55 PG_SHDEPEND.....	635
18.2.56 PG_SHDESCRIPTION.....	636
18.2.57 PG_SHSECLABEL.....	637
18.2.58 PG_STATISTIC.....	637
18.2.59 PG_STATISTIC_EXT.....	639
18.2.60 PG_SYNONYM.....	640
18.2.61 PG_TABLESPACE.....	641
18.2.62 PG_TRIGGER.....	641
18.2.63 PG_TS_CONFIG.....	642
18.2.64 PG_TS_CONFIG_MAP.....	643
18.2.65 PG_TS_DICT.....	643
18.2.66 PG_TS_PARSER.....	644
18.2.67 PG_TS_TEMPLATE.....	645
18.2.68 PG_TYPE.....	645
18.2.69 PG_USER_MAPPING.....	649
18.2.70 PG_USER_STATUS.....	650
18.2.71 PG_WORKLOAD_GROUP.....	650
18.2.72 PGXC_CLASS.....	650
18.2.73 PGXC_GROUP.....	651
18.2.74 PGXC_NODE.....	652
18.3 System Views.....	653
18.3.1 ALL_ALL_TABLES.....	653
18.3.2 ALL_CONSTRAINTS.....	653
18.3.3 ALL_CONS_COLUMNS.....	654
18.3.4 ALL_COL_COMMENTS.....	654
18.3.5 ALL_DEPENDENCIES.....	655
18.3.6 ALL_IND_COLUMNS.....	655
18.3.7 ALL_IND_EXPRESSIONS.....	656
18.3.8 ALL_INDEXES.....	656
18.3.9 ALL_OBJECTS.....	657
18.3.10 ALL_PROCEDURES.....	657
18.3.11 ALL_SEQUENCES.....	658
18.3.12 ALL_SOURCE.....	658
18.3.13 ALL_SYNONYMS.....	659
18.3.14 ALL_TAB_COLUMNS.....	659
18.3.15 ALL_TAB_COMMENTS.....	660

18.3.16 ALL_TABLES.....	660
18.3.17 ALL_USERS.....	661
18.3.18 ALL_VIEWS.....	661
18.3.19 DBA_DATA_FILES.....	662
18.3.20 DBA_USERS.....	662
18.3.21 DBA_COL_COMMENTS.....	662
18.3.22 DBA_CONSTRAINTS.....	662
18.3.23 DBA_CONS_COLUMNS.....	663
18.3.24 DBA_IND_COLUMNS.....	663
18.3.25 DBA_IND_EXPRESSIONS.....	664
18.3.26 DBA_IND_PARTITIONS.....	664
18.3.27 DBA_INDEXES.....	665
18.3.28 DBA_OBJECTS.....	666
18.3.29 DBA_PART_INDEXES.....	666
18.3.30 DBA_PART_TABLES.....	667
18.3.31 DBA_PROCEDURES.....	668
18.3.32 DBA_SEQUENCES.....	668
18.3.33 DBA_SOURCE.....	668
18.3.34 DBA_SYNONYMS.....	669
18.3.35 DBA_TAB_COLUMNS.....	669
18.3.36 DBA_TAB_COMMENTS.....	670
18.3.37 DBA_TAB_PARTITIONS.....	671
18.3.38 DBA_TABLES.....	671
18.3.39 DBA_TABLESPACES.....	672
18.3.40 DBA_TRIGGERS.....	672
18.3.41 DBA_VIEWS.....	673
18.3.42 DUAL.....	673
18.3.43 GLOBAL_WORKLOAD_SQL_COUNT.....	673
18.3.44 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME.....	674
18.3.45 GS_ALL_CONTROL_GROUP_INFO.....	675
18.3.46 GS_CLUSTER_RESOURCE_INFO.....	675
18.3.47 GS_INSTR_UNIQUE_SQL.....	676
18.3.48 GS_SESSION_CPU_STATISTICS.....	680
18.3.49 GS_SESSION_MEMORY_STATISTICS.....	681
18.3.50 GS_SQL_COUNT.....	681
18.3.51 GS_WLM_CGROUP_INFO.....	683
18.3.52 GS_WLM_OPERATOR_HISTORY.....	684
18.3.53 GS_WLM_OPERATOR_STATISTICS.....	684
18.3.54 GS_WLM_SESSION_HISTORY.....	685
18.3.55 GS_WLM_SESSION_STATISTICS.....	689
18.3.56 GS_WLM_WORKLOAD_RECORDS.....	692
18.3.57 PGXC_WORKLOAD_SQL_COUNT.....	693

18.3.58 PGXC_WORKLOAD_SQL_ELAPSE_TIME.....	693
18.3.59 GS_WORKLOAD_TRANSACTION.....	694
18.3.60 GS_STAT_DB_CU.....	695
18.3.61 GS_STAT_SESSION_CU.....	695
18.3.62 GS_TOTAL_NODEGROUP_MEMORY_DETAIL.....	696
18.3.63 GS_USER_TRANSACTION.....	696
18.3.64 PG_AVAILABLE_EXTENSION_VERSIONS.....	697
18.3.65 PG_AVAILABLE_EXTENSIONS.....	698
18.3.66 PG_COMM_CLIENT_INFO.....	698
18.3.67 PG_COMM_DELAY.....	699
18.3.68 PG_COMM_STATUS.....	699
18.3.69 PG_COMM_RECV_STREAM.....	700
18.3.70 PG_COMM_SEND_STREAM.....	701
18.3.71 PG_CONTROL_GROUP_CONFIG.....	702
18.3.72 PG_CURSORS.....	702
18.3.73 PG_EXT_STATS.....	703
18.3.74 PG_GET_INVALID_BACKENDS.....	705
18.3.75 PG_GET_SENDERS_CATCHUP_TIME.....	706
18.3.76 PG_GROUP.....	706
18.3.77 PG_INDEXES.....	706
18.3.78 PG_LOCKS.....	707
18.3.79 PG_NODE_ENV.....	708
18.3.80 PG_OS_THREADS.....	709
18.3.81 PG_POOLER_STATUS.....	709
18.3.82 PG_PREPARED_STATEMENTS.....	710
18.3.83 PG_PREPARED_XACTS.....	711
18.3.84 PG_REPLICATION_SLOTS.....	711
18.3.85 PG_ROLES.....	712
18.3.86 PG_RULES.....	713
18.3.87 PG_RUNNING_XACTS.....	714
18.3.88 PG_SECLABELS.....	714
18.3.89 PG_SESSION_WLMSTAT.....	715
18.3.90 PG_SESSION_IOSTAT.....	717
18.3.91 PG_SETTINGS.....	718
18.3.92 PG_SHADOW.....	719
18.3.93 PG_SHARED_MEMORY_DETAIL.....	720
18.3.94 PG_STATS.....	721
18.3.95 PG_STAT_ACTIVITY.....	723
18.3.96 PG_STAT_ALL_INDEXES.....	726
18.3.97 PG_STAT_ALL_TABLES.....	727
18.3.98 PG_STAT_BAD_BLOCK.....	728
18.3.99 PG_STAT_BGWRITER.....	728

18.3.100 PG_STAT_DATABASE.....	729
18.3.101 PG_STAT_DATABASE_CONFLICTS.....	731
18.3.102 PG_STAT_GET_MEM_MBYTES_RESERVED.....	731
18.3.103 PG_STAT_USER_FUNCTIONS.....	732
18.3.104 PG_STAT_USER_INDEXES.....	732
18.3.105 PG_STAT_USER_TABLES.....	733
18.3.106 PG_STAT_REPLICATION.....	734
18.3.107 PG_STAT_SYS_INDEXES.....	735
18.3.108 PG_STAT_SYS_TABLES.....	736
18.3.109 PG_STAT_XACT_ALL_TABLES.....	737
18.3.110 PG_STAT_XACT_SYS_TABLES.....	738
18.3.111 PG_STAT_XACT_USER_FUNCTIONS.....	738
18.3.112 PG_STAT_XACT_USER_TABLES.....	739
18.3.113 PG_STATIO_ALL_INDEXES.....	739
18.3.114 PG_STATIO_ALL_SEQUENCES.....	740
18.3.115 PG_STATIO_ALL_TABLES.....	740
18.3.116 PG_STATIO_SYS_INDEXES.....	741
18.3.117 PG_STATIO_SYS_SEQUENCES.....	741
18.3.118 PG_STATIO_SYS_TABLES.....	742
18.3.119 PG_STATIO_USER_INDEXES.....	743
18.3.120 PG_STATIO_USER_SEQUENCES.....	743
18.3.121 PG_STATIO_USER_TABLES.....	743
18.3.122 PG_THREAD_WAIT_STATUS.....	744
18.3.123 PG_TABLES.....	757
18.3.124 PG_TIMEZONE_ABBREVS.....	757
18.3.125 PG_TIMEZONE_NAMES.....	758
18.3.126 PG_TOTAL_MEMORY_DETAIL.....	758
18.3.127 PG_TOTAL_USER_RESOURCE_INFO.....	759
18.3.128 PG_USER.....	760
18.3.129 PG_USER_MAPPINGS.....	761
18.3.130 PG_VIEWS.....	762
18.3.131 PG_WLM_STATISTICS.....	762
18.3.132 PGXC_COMM_CLIENT_INFO.....	763
18.3.133 PGXC_COMM_DELAY.....	764
18.3.134 PGXC_COMM_RECV_STREAM.....	765
18.3.135 PGXC_COMM_SEND_STREAM.....	765
18.3.136 PGXC_COMM_STATUS.....	766
18.3.137 PGXC_GET_STAT_ALL_TABLES.....	767
18.3.138 PGXC_GET_TABLE_SKEWNESS.....	768
18.3.139 PGXC_GTM_SNAPSHOT_STATUS.....	769
18.3.140 PGXC_INSTR_UNIQUE_SQL.....	769
18.3.141 PGXC_NODE_ENV.....	769

18.3.142 PGXC_OS_THREADS.....	770
18.3.143 PGXC_PREPARED_XACTS.....	770
18.3.144 PGXC_RUNNING_XACTS.....	770
18.3.145 PGXC_STAT_ACTIVITY.....	771
18.3.146 PGXC_STAT_BAD_BLOCK.....	773
18.3.147 PGXC_SQL_COUNT.....	774
18.3.148 PGXC_THREAD_WAIT_STATUS.....	774
18.3.149 PGXC_TOTAL_MEMORY_DETAIL.....	776
18.3.150 PGXC_VARIABLE_INFO.....	777
18.3.151 PGXC_WLM_OPERATOR_HISTORY.....	778
18.3.152 PGXC_WLM_OPERATOR_INFO.....	778
18.3.153 PGXC_WLM_OPERATOR_STATISTICS.....	778
18.3.154 PGXC_WLM_SESSION_INFO.....	779
18.3.155 PGXC_WLM_SESSION_HISTORY.....	779
18.3.156 PGXC_WLM_SESSION_STATISTICS.....	779
18.3.157 PGXC_WLM_WORKLOAD_RECORDS.....	779
18.3.158 GS_WORKLOAD_SQL_COUNT.....	780
18.3.159 GS_WORKLOAD_SQL_ELAPSE_TIME.....	781
18.3.160 PLAN_TABLE.....	782
18.3.161 PLAN_TABLE_DATA.....	783
18.3.162 PV_FILE_STAT.....	784
18.3.163 PV_INSTANCE_TIME.....	785
18.3.164 PV_OS_RUN_INFO.....	785
18.3.165 PV_SESSION_MEMORY.....	785
18.3.166 PV_SESSION_MEMORY_DETAIL.....	786
18.3.167 PV_SESSION_STAT.....	787
18.3.168 PV_SESSION_TIME.....	787
18.3.169 PV_TOTAL_MEMORY_DETAIL.....	787
18.3.170 REDACTION_COLUMNS.....	788
18.3.171 REDACTION_POLICIES.....	789
18.3.172 USER_COL_COMMENTS.....	790
18.3.173 USER_CONSTRAINTS.....	790
18.3.174 USER_CONS_COLUMNS.....	791
18.3.175 USER_INDEXES.....	791
18.3.176 USER_IND_COLUMNS.....	792
18.3.177 USER_IND_EXPRESSIONS.....	792
18.3.178 USER_IND_PARTITIONS.....	793
18.3.179 USER_JOBS.....	793
18.3.180 USER_OBJECTS.....	794
18.3.181 USER_PART_INDEXES.....	795
18.3.182 USER_PART_TABLES.....	796
18.3.183 USER_PROCEDURES.....	796

18.3.184 USER_SEQUENCES.....	797
18.3.185 USER_SOURCE.....	797
18.3.186 USER_SYNONYMS.....	797
18.3.187 USER_TAB_COLUMNS.....	798
18.3.188 USER_TAB_COMMENTS.....	799
18.3.189 USER_TAB_PARTITIONS.....	799
18.3.190 USER_TABLES.....	800
18.3.191 USER_TRIGGERS.....	800
18.3.192 USER_VIEWS.....	801
18.3.193 V\$SESSION.....	801
18.3.194 V\$SESSION_LONGOPS.....	801
19 Information Schema.....	803
20 Tool Reference.....	804
20.1 gs_dump.....	804
20.2 gs_dumpall.....	816
20.3 gs_restore.....	821
21 Glossary.....	829
A Change History.....	844

1 Welcome

1.1 Target Readers

This document is intended for database designers, application developers, and database administrators, and provides information required for designing, building, querying and maintaining data warehouses.

As a database administrator or application developer, you need to be familiar with:

- Knowledge about OSs, which is the basis for everything.
- SQL syntax, which is the necessary skill for database operation.

Statement

When writing documents, the writers of GaussDB(DWS) try their best to provide guidance from the perspective of commercial use, application scenarios, and task completion. Even so, references to PostgreSQL content may still exist in the document. For this type of content, the following PostgreSQL Copyright is applicable:

Postgres-XC is Copyright © 1996-2013 by the PostgreSQL Global Development Group.

PostgreSQL is Copyright © 1996-2013 by the PostgreSQL Global Development Group.

Postgres95 is Copyright © 1994-5 by the Regents of the University of California.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND THE UNIVERSITY OF

CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

1.2 Reading Guide

If you are a new GaussDB(DWS) user, you are advised to read the following contents first:

- **Service Overview:** Describes the features, functions, and application scenarios of GaussDB(DWS).
- **Getting Started:** The "Getting Started" section of GaussDB(DWS) guides you through creating a data warehouse cluster, creating a database table, uploading data, and testing queries.

If you intend to or are migrating applications from other data warehouses to GaussDB(DWS), you might want to know how GaussDB(DWS) differs from them.

GaussDB(DWS), during the development of the database application, provides the following table to help you find wanted information.

If you want to...	Query Suggestions
Quickly get started with GaussDB(DWS).	First, follow the steps in Data Warehouse Service Getting Started to quickly deploy a cluster, connect to a database, and try some queries. When you are ready to construct a database, load data to tables and compile the query content to operate the data in the data warehouse. Then, you can return to <i>Database Developer Guide</i> .
Understand the internal architecture of a GaussDB(DWS) data warehouse.	Product Architecture describes the architecture of GaussDB(DWS). To know more about GaussDB(DWS), go to the GaussDB(DWS) home page.
Learn how to design tables to achieve the excellent performance.	Development and Design Proposal introduces the design specifications that should be complied with during the development of database applications. Modeling compliant with these specifications fits the distributed processing architecture of GaussDB(DWS) and provides efficient SQL code. To facilitate service execution through optimization, you can refer to Query Performance Tuning . Successful performance optimization depends more on database administrators' experience and judgment than on instructions and explanation. However, Query Performance Tuning still tries to systematically illustrate the performance optimization methods for application development personnel and new GaussDB(DWS) database administrators.

If you want to...	Query Suggestions
Load data.	Data Import describes how to import data to GaussDB(DWS). Excellent Practices for Data Import provides key points for quick data import.
Manage users, groups, and database security.	Database Security Management covers topics on database security.
Monitor and optimize system performance.	Overview of System Catalogs and System Views describes the system catalogs and views where you can query for the database status and monitor the query content and process. You should also learn from Management Guide about how to use the GaussDB(DWS) management console to check the system status and monitor metrics.

1.3 Conventions

Example Conventions

Example	Description
dbadmin	Indicates the user operating and maintaining GaussDB(DWS) appointed when the cluster is created.
8000	Indicates the port number used by GaussDB(DWS) to monitor connection requests from the client.

SQL examples in this manual are developed based on the TPC-DS model. Before you execute the examples, install the TPC-DS benchmark by following the instructions on the official website <http://www.tpc.org/tpcds/>.

SQL Syntax Text Conventions

To better understand the syntax usage, you can refer to the SQL syntax text conventions described as follows:

Format	Description
Uppercase characters	Keywords must be in uppercase.
Lowercase characters	Indicates that parameters must be in lowercase.
[]	Indicates that the items in brackets [] are optional.

Format	Description
...	Indicates that preceding elements can appear repeatedly.
[x y ...]	Indicates that one item is selected from two or more options or no item is selected.
{ x y ... }	Indicates that one item is selected from two or more options.
[x y ...] [...]	Indicates that multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with spaces.
[x y ...] [, ...]	Indicates that multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with commas (,).
{ x y ... } [...]	Indicates that at least one parameter can be selected. If multiple parameters are selected, separate them with spaces.
{ x y ... } [, ...]	Indicates that at least one parameter can be selected. If multiple parameters are selected, separate them with commas (,).

1.4 Prerequisites

Complete the following tasks before you perform operations described in this document:

- Create a GaussDB(DWS) cluster.
- Install an SQL client.
- Connect the SQL client to the default database of the cluster.

For detailed instructions on the above tasks, see [Data Warehouse Service Getting Started](#).

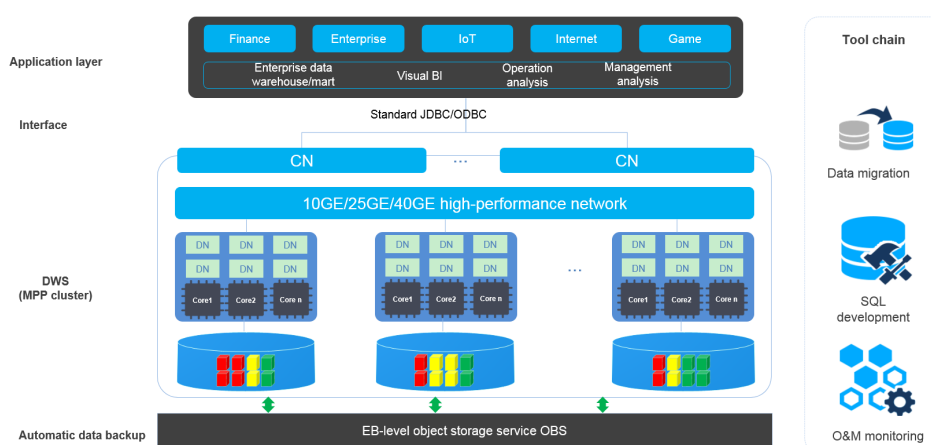
You should also know how to use the SQL client and have a basic understanding of the SQL language.

2 System Overview

2.1 Product Architecture

GaussDB(DWS) is a distributed parallel database cluster with the shared-nothing architecture, as shown in [Figure 2-1](#).

Figure 2-1 GaussDB(DWS) product architecture



- **Application layer**
Data loading tools, Extract-Transform-Load (ETL) tools, Business Intelligence (BI) tools, and data mining and analysis tools can be integrated with GaussDB(DWS) through standard interfaces. GaussDB(DWS) is compatible with PostgreSQL and the SQL syntax is modified to make it compatible with Oracle and Teradata. Applications can be smoothly migrated to GaussDB(DWS) with few changes.
- **Interface**
Applications can connect to GaussDB(DWS) through standard JDBC 4.0 and ODBC 3.5.
- **GaussDB(DWS) (MPP cluster)**

The cluster consists of modules used for data management. [Figure 2-2](#) shows these modules and [Table 2-1](#) describes their functions.

- Automatic data backup

Cluster snapshots can be automatically backed up to Object Storage Service (OBS), an EB-level object storage service, which facilitates periodic backup of the cluster during off-peak hours, ensuring data recovery after a cluster exception occurs.

A snapshot is a complete backup of GaussDB(DWS) at a specified time point. It records all configuration data and service data of the cluster at the specified moment.

- Tool chain

The parallel data loading tool General Data Service (GDS), syntax migration tool DSC, and SQL development tool Data Studio are provided. The cluster O&M can be monitored through a console.

[Figure 2-2](#) shows the logical architecture of GaussDB(DWS). For details, see [Table 2-1](#).

Figure 2-2 Cluster logical architecture

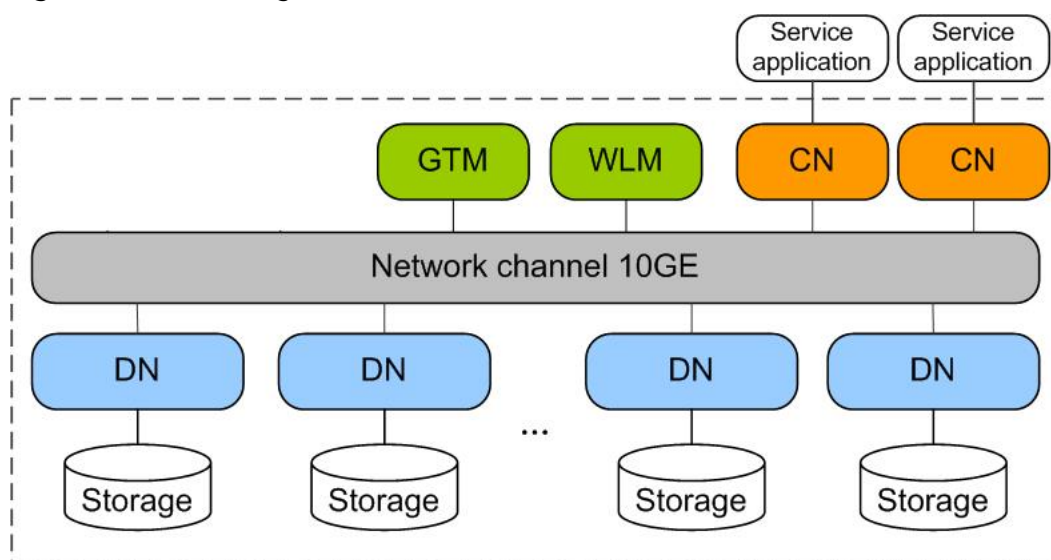


Table 2-1 Cluster architecture description

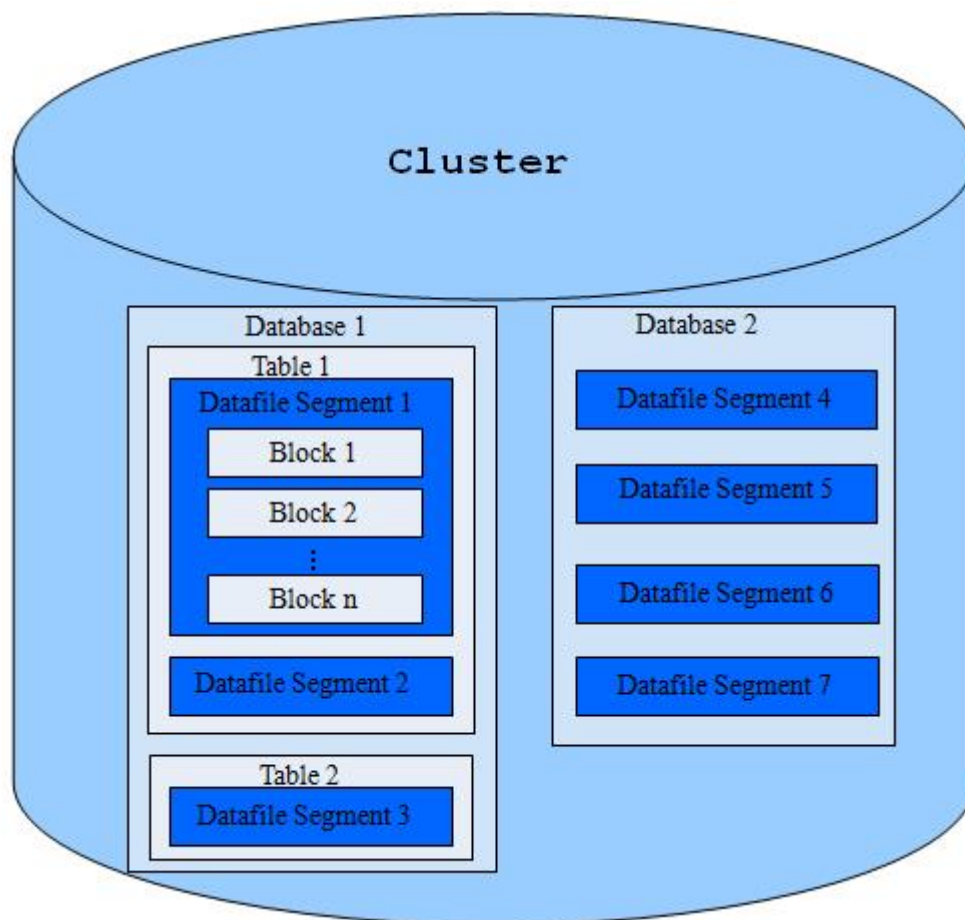
Name	Description
GTM	Global Transaction Manager: generates and maintains the global unique information, such as the global transaction ID, transaction snapshot, and timestamp.
WLM	Workload Manager: controls allocation of system resources to prevent service congestion and system crash resulting from excessive workload.

Name	Description
CN	Coordinator: receives access requests from applications, and returns execution results to the client. The CN breaks down tasks and allocates task fragments to different DNs for parallel processing.
DN	Datanode: stores service data by column or row or in the hybrid mode, executes data query tasks, and returns execution results to CNs.
Storage	Functions as the server's local storage resources to store data permanently.

DNs in a cluster store data on disks. [Figure 2-3](#) logically describes the objects on each DN and the relationship among them.

- Database: A database manages various data objects and is isolated from each other.
- Data file Segment: A data file, each of which stores data of only one table. A table containing more than 1 GB of data is stored in multiple data file segments.
- Table: One table belongs to only one database.
- Block: The basic unit of database management. Its default size is 8 KB.

Data can be distributed in REPLICATION, ROUNDROBIN, or HASH mode. You can set it while creating a table. ROUNDROBIN only applies to foreign tables.

Figure 2-3 Database logical architecture

2.2 HA Transaction Processing

GaussDB(DWS) manages cluster transactions, the basis of HA and failovers. This ensures speedy fault recovery, guarantees the Atomicity, Consistency, Isolation, Durability (ACID) properties for transactions and after a recovery, and enables concurrent control.

Fault Rectification

GaussDB(DWS) provides an HA mechanism to reduce the service interruption time when a cluster is faulty. It protects key user programs to continuously provide external services, minimizing the impact of hardware, software, and human faults on services and ensuring service continuity.

- Hardware HA: Disk RAID, switch stacking, NIC bond, and uninterruptible power supply (UPS)
- Software HA: HA mechanism used for instances in the GaussDB(DWS) cluster, such as CNs, GTMs, and DNPs)

Transaction Management

- Transaction blocks are supported. You can run **start transaction** to make the startup of a transaction block explicit.

- Single-statement transactions are supported. If you do not explicitly start a transaction, a single statement is processed as a transaction.
- Distributed transaction management and global transaction information management are supported. This includes gxid, snapshot, timestamp management, distributed transaction status management, and gxid overflow processing.
- Distributed transactions have ACID properties.
- Deadlocks are prevented in the distributed system. A transaction will be unlocked immediately after a deadlock (if any).

2.3 High Query Performance

The following GaussDB(DWS) features help achieve high query performance.

Fully Parallel Query

GaussDB(DWS) is an MPP system with the shared-nothing architecture. It consists of multiple independent logical nodes that do not share system resources, such as the CPU, memory, and storage units. In this system architecture, service data is separately stored on numerous nodes. Data analysis task is executed at the location nearest the data. Massively parallel data processing enables quick response.

In addition, GaussDB(DWS) improves data query performance by executing operators in parallel, executing commands in registers in parallel, and using LLVM to dynamically compile the logical conditions of redundancy prune.

Hybrid Row-Column Storage

GaussDB(DWS) supports both the row and column storage models. You can choose a row- or column-store table as needed.

The hybrid row/column-based storage engine provides users with a better data compression ratio (column-based storage), index performance (column-based storage), and point update and point query (row-based storage) performance.

Data Compression in Column Storage

You can compress old, inactive data to free up space, reducing procurement and O&M costs.

In GaussDB(DWS), data can be compressed using the following Delta Value Encoding, Dictionary, RLE, LZ4, and ZLIB algorithms. The system automatically selects a compression algorithm based on data characteristics. The average compression ratio is 7:1. Compressed data can be directly accessed and is transparent to services, greatly reducing the preparation time before accessing historical data.

2.4 Technical Specifications

Table 2-2 Technical specifications

Item	Maximum Value
Data capacity	10PB
Number of cluster nodes	256
Size of a single table	1PB
Size of data in each row	1 GB
Size of a single field in each record	1 GB
Number of records in each table	2 ⁵⁵
Number of columns in each table	1600
Number of indexes in each table	Unlimited
Number of columns contained in the index of each table	32
Number of constraints in each table	Unlimited
Number of concurrent connections	60 for analytical long transactions; 5000 for short transactions.
Number of partitions of the partitioned table	32768
Size of each single partition in a partitioned table	1 PB
Number of records in each partition of a partitioned table	2 ⁵⁵

2.5 SQL Syntax

Structured Query Language (SQL) is the standard language on relational databases (RDB). SQL is a universal and useful RDB language.

GaussDB(DWS) supports most SQL standards.

GaussDB(DWS) has the following constraints in terms of the compatibility with SQL standards:

- The **SERIALIZABLE** transaction isolation level is not supported.
- The **PREPARE** statement does not support complex queries.
- The row-level trigger and **COPY** cannot be used together.
- **CREATE TABLE AS EXECUTE** cannot be used.
- **WHERE CURRENT OF** cannot be used.
- A cursor does not support **MOVE BACKWARD**.
- **FOREIGN DATA WRAPPER** cannot be used.
- **LISTEN**, **UNLISTEN**, and **NOTIFY** are supported only on CNs.
- **SECURITY LABEL** cannot be used.
- The distribution columns in a table cannot be updated.

For details about the error messages during SQL execution, see [Error Code Reference](#).

2.6 Related Concepts

Instance

In GaussDB(DWS), instances are a group of database processes running in the memory. An instance can manage one or more databases that form a cluster. A cluster is an area in the storage disk. This area is initialized during installation and composed of a directory. The directory, called data directory, stores all data and is created by **initdb**. Theoretically, one server can start multiple instances on different ports, but GaussDB(DWS) manages only one instance at a time. The start and stop of an instance rely on the specific data directory. For compatibility purposes, the concept of instance name may be introduced.

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** by default. Objects managed by a database can be distributed to multiple tablespaces.

Schema

GaussDB(DWS) schemas logically separate databases. All database objects are created under certain schemas. In GaussDB(DWS), 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(DWS) 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(DWS), the difference between roles and users is that a role does not have the LOGIN permission by default. In GaussDB(DWS), 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 Management

In GaussDB(DWS), transactions are managed by multi-version concurrency control (MVCC) and two-phase locking (2PL). It enables smooth data reads and writes. In GaussDB(DWS), MVCC saves historical version data together with the current tuple version. GaussDB(DWS) uses the VACUUM process instead of rollback segments to routinely delete historical version data. Unless in performance optimization, you do not need to pay attention to the VACUUM process. Transactions are automatically submitted in GaussDB(DWS).

3 Syntax Compatibility Differences Between Teradata and Oracle

GaussDB(DWS) is compatible with Teradata and Oracle syntax, of which the syntax behavior is different.

Table 3-1 Compatibility differences

Compatibility Item	Teradata	Oracle
Data type DATE	Stores year, month, and day values.	Converts the DATE data type to the TIMESTAMP data type which stores year, month, day, hour, minute, and second values.
Empty string	An empty string is distinguished from a NULL value.	An empty string is treated as a NULL value.
Conversion from an empty string to INT	0	NULL
Automatic truncation of overlong characters	Supported (only if td_compatible_truncation is set to on)	Not supported
VARCHAR + INT calculation	Converts to NUMERIC + NUMERIC calculation.	Converts to BIGINT + INT calculation.

Compatibility Item	Teradata	Oracle
CASE and COALESCE expressions	Is compatible with behavior of Teradata and supports type conversion between digits and strings. For example, if input parameters for COALESCE are of INT and VARCHAR types, the parameters are resolved as VARCHAR type.	Reports error.

4 Database Security Management

4.1 Managing Users and Their Permissions

4.1.1 Default Permission Mechanism

A user who creates an object is the owner of this object. By default, [Separation of Duty](#) 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(DWS) supports the following permissions: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCES**, **CREATE**, **CONNECT**, **EXECUTE**, and **USAGE**. Permission types are associated with object types. For permission details, see **GRANT**.

To remove permissions, use **REVOKE**. Object owner permissions such as **ALTER**, **DROP**, **GRANT**, and **REVOKE** are implicit and cannot be granted or revoked. That is, you have the implicit permissions for an object if you are the owner of 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**.

4.1.2 System Administrator

A system administrator is an account with the **SYSADMIN** permission. After a cluster is installed, a system administrator has the permissions of all object owners by default.

The user **GaussDB(DWS)** created upon GaussDB(DWS) startup is a system administrator.

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

```
CREATE USER sysadmin WITH SYSADMIN password "Bigdata@123";
```

Alternatively, you can run the following statement:

```
ALTER USER joe SYSADMIN;
```

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

4.1.3 Separation of Duty

Descriptions in [Default Permission Mechanism](#) and [System 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-duty to delegate system administrator permissions to security administrators and audit administrators.

After separation of duty is enabled, a system administrator does not have the **CREATEROLE** attribute (security administrator) and **AUDITADMIN** attribute (audit administrator). That is, you do not have the permissions to create roles and users and the permissions to view and maintain database audit logs. For details about the **CREATEROLE** and **AUDITADMIN** attributes, see **CREATE ROLE**.

After separation-of-duty is enabled, system administrators have the permissions only for the objects owned by them.

For details about how to enable separation-of-duty, see [Separating Rights of Roles](#).

For details about permission changes before and after enabling separation-of-duty, see [Table 4-1](#) and [Table 4-2](#).

Table 4-1 Default user permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespace	Can create, modify, delete, access, and allocate tablespaces.	Cannot create, modify, delete, or allocate tablespaces, with authorization required for accessing tablespaces.		
Table	Has permissions for all tables.	Has permissions for its own tables, but does not have permissions for other users' tables.		

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Index	Can create indexes on all tables.	Can create indexes on their own tables.		
Schema	Has permissions for all schemas.	Has all permissions for its own schemas, but does not have permissions for other users' schemas.		
Function	Has permissions for all functions.	Has permissions for its own functions, has the call permission for other users' functions in the public schema, but does not have permissions for other users' functions in other schemas.		
Customized view	Has permissions for all views.	Has permissions for its own views, but does not have permissions for other users' views.		
System catalog and system view	Has permissions to query all system catalogs and views.	Has permissions to query only some system catalogs and views. For details, see System Catalogs and System Views .		

Table 4-2 Changes in permissions after separation-of-duty

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Database	No change	No change		
Table	Permissions reduced Has all permissions for its own tables, but does not have permissions for other users' tables in their schemas.	No change		
Index	Permissions reduced Can create indexes on its own tables.	No change		
Schema	Permissions reduced Has all permissions for its own schemas, but does not have permissions for other users' schemas.	No change		

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Function	Permissions reduced Has all permissions for its own functions, but does not have permissions for other users' functions in their schemas.	No change		
Customized view	Permissions reduced Has all permissions for its own views and other users' views in the public schema, but does not have permissions for other users' views in their schemas.	No change		
System catalog and system view	No change	No change	No change	Has no permission to view any system catalogs or views.

4.1.4 Users

You can use **CREATE USER** and **ALTER USER** to create and manage database users, respectively. The database cluster has one or more named 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-duty** scenarios, a GaussDB(DWS) user account can be created and deleted only by a system administrator or a security administrator with the **CREATEROLE** attribute. In separation-of-duty scenarios, a user account can be created only by a security administrator.

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

Adding, Modifying, and Deleting Users

- To create a user, use the SQL **CREATE USER** statement.
For example, create a user **joe** and set the **CREATEDB** attribute for the user.
CREATE USER joe WITH CREATEDB PASSWORD "Bigdata@123";
CREATE USER
- To create a system administrator, use the **CREATE USER** statement with the **SYSADMIN** parameter.

- To delete an existing user, use the DROP USER statement.
- To change a user account (for example, rename the user or change the password), use the ALTER USER statement.
- To view a user list, query the **PG_USER** view.

```
SELECT * FROM pg_user;
```
- To view user attributes, query the system catalog **PG_AUTHID**.

```
SELECT * FROM pg_authid;
```

Private Users

If multiple service departments use different database user accounts to perform service operations and a database maintenance department at the same level uses database administrator accounts to perform maintenance operations, service departments may require that database administrators, without specific authorization, can manage (**DROP**, **ALTER**, and **TRUNCATE**) their data but cannot access (**INSERT**, **DELETE**, **UPDATE**, **SELECT**, and **COPY**) the data. That is, the management permissions of database administrators for tables need to be isolated from their access permissions to improve the data security of common users.

In **Separation of Duty** mode, a database administrator does not have permissions for the tables in schemas of other users. In this case, database administrators have neither management permissions nor access permissions, which does not meet the requirements of the service departments mentioned above. Therefore, GaussDB(DWS) provides private users to solve the problem. That is, create private users with the **INDEPENDENT** attribute in non-separation-of-duties mode.

```
CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "1234@abc";
```

Database administrators can manage (**DROP**, **ALTER**, and **TRUNCATE**) objects of private users but cannot access (**INSERT**, **DELETE**, **SELECT**, **UPDATE**, **COPY**, **GRANT**, **REVOKE**, and **ALTER OWNER**) the objects before being authorized.

4.1.5 Roles

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

GaussDB(DWS) 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 non-**separation-of-duty** scenarios, a role can be created, modified, and deleted only by a system administrator or a user with the **CREATEROLE** attribute. In

separation-of-duty scenarios, a role can be created, modified, and deleted only by 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 member users in the role.

4.1.6 Schema

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 schema named **public** is created by default, and all users have permissions for this schema. You can group database objects by schema. A schema is similar to an OS directory but cannot be nested.

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 a user is created in the initial database **postgres**, the system automatically creates a schema with the same name as the user. In other databases, such a schema needs to be manually created.

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**. Any user can create a schema.
- To change the name or owner of a schema, use **ALTER SCHEMA**. Only the schema owner can do so.
- To delete a schema and its objects, use **DROP SCHEMA**. Only the schema owner can do so.
- 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.

```
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**.

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

```
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**.

```
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 **search_path** to **myschema** and **public** (**myschema** will be searched first):

```
SET SEARCH_PATH TO myschema, public;  
SET
```

4.1.7 User Permission Setting

- To grant the permission for an object directly 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 the names of the objects but cannot actually access them.

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

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

- Granting a role to a user allows the user to inherit the object permissions of the role.

- Create a role.

Create a role **lily** and grant the system permission **CREATEDB** to the role.

```
CREATE ROLE lily WITH CREATEDB PASSWORD "Bigdata@123";
```

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

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

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

- c. Grant the role permissions to a user.

```
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**.

4.1.8 Row-Level Access Control

The row-level access control 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 access control 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 a SQL statement meets the specified row-level access control 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 access control 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.
CREATE ROLE alice PASSWORD 'Gauss@123';
CREATE ROLE bob PASSWORD 'Gauss@123';
CREATE ROLE peter PASSWORD 'Gauss@123';

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

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

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

-- Enable row-level access control.
ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

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

-- View table details.
\d+ 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, compression=no, enable_rowsecurity=true  
  
-- Switch to user alice and run SELECT * FROM public.all_data.  
SELECT * FROM public.all_data;  
id | role | data  
-----+-----  
 1 | alice | alice data  
(1 row)  
  
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.  
SELECT * FROM public.all_data;  
id | role | data  
-----+-----  
 3 | peter | peter data  
(1 row)  
  
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)
```

4.1.9 Data Redaction

Database permission control in GaussDB(DWS) can be implemented at the level of column and row. In the current database permission system, if sensitive information needs to be hidden for a user, only the permission on the table or column can be revoked. For unauthorized query, an error is reported.

GaussDB(DWS) supports data masking to protect sensitive information with higher availability. You can create a data redaction policy for a table object. A data redaction policy is a combination of specific database users, data objects, SQL operations, and redaction modes. The data redaction feature can be used to hide sensitive information of some users, ensuring sensitive information security.

Data redaction aims to prevent privacy leakage of some data columns in a table. After a column object is bound to a redaction policy, the redaction policy that meets the conditions is applied to the execution plan in the query rewriting phase, affecting the final execution result of the **SELECT** statement.

To ensure data security, an error message is displayed when the **INSERT INTO SELECT**, **UPDATE FROM**, or **MERGE INTO** statement contains sensitive information.

GaussDB(DWS) provides function interfaces for creating, modifying, and deleting redaction policies. For details, see the data redaction function.

Scenario example: A table summarizes personal information. Many attributes in the personal information are sensitive data. A redaction policy is created for sensitive columns.

Create users **alice**, **bob**, and **peter**.

```
CREATE ROLE alice PASSWORD 'Gauss@123';
CREATE ROLE bob PASSWORD 'Gauss@123';
CREATE ROLE peter PASSWORD 'Gauss@123';
```

User **alice** creates the **customer** table in public mode to record user information (name and salary).

```
CREATE TABLE public.customer(id int, name varchar(100), salary int);
```

Insert three users to the data table.

```
INSERT INTO customer VALUES(1, 'Michael', 10000);
INSERT INTO customer VALUES(2, 'bryant', 20000);
INSERT INTO customer VALUES(3, 'harden', 15000);
```

User **alice** grants the read permission of the **customer** table to **bob** and **peter**.

```
GRANT SELECT ON customer TO bob, peter;
```

alice does not want **bob** and **peter** to view salary data, and adds a redaction policy to the **customer** table.

```
SELECT
DBMS_REDACT.add_policy(object_schema=>'public',object_name=>'customer',policy_name=>'pol_1',column_
name=>'salary',expression=>'1=1');
```

After the user is switched to **bob** or **peter**, the salary data in the **customer** table cannot be viewed.

```
set role bob password 'Gauss@123';
SET
select * from customer;
id | name | salary
-----+-----
1 | Michael |
2 | bryant |
3 | harden |
(3 rows)

set role peter password 'Gauss@123';
SET
select * from customer;
id | name | salary
-----+-----
1 | Michael |
2 | bryant |
3 | harden |
(3 rows)
```

alice changes her mind and does not want **peter** to see salary data, and modifies the policy.

```
SELECT
DBMS_REDACT.alter_policy(object_schema=>'public',object_name=>'customer',policy_name=>'pol_1',action=
>'3',expression=>'SYS_CONTEXT("USERENV", "CURRENT_USER") = "peter"');
```

Switch to user **bob** and view the salary data in the **customer** table.

```
set role bob password 'Gauss@123';
SET
SELECT * FROM customer;
id | name | salary
```

```

+-----+-----+
1 | Michael | 10000
2 | bryant | 20000
3 | harden | 15000
(3 rows)

```

View the existing redaction policy and redacted column information.

```

select * from redaction_policies;
object_schema | object_owner | object_name | policy_name |
expression          | enable | policy_description
+-----+-----+-----+-----+-----+
public        | yxw        | customer   | pol_1       | (sys_context('USERENV':text, 'CURRENT_USER':text) =
'peter':text) | t       |
(1 row)

select * from redaction_columns;
object_schema | object_owner | object_name | column_name | function_type | function_parameters |
regexp_pattern | regexp_replace_string | regexp_position | regexp_occurrence | regexp_match_parameter |
column
n_description
+-----+-----+-----+-----+-----+-----+
public        | yxw        | customer   | salary      | 1             |                    |
(1 row)

```

4.2 Setting Account Security Policies

4.2.1 Setting Account Security Locking Policies

Context

For data security purposes, GaussDB(DWS) provides a series of security measures, such as automatically locking and unlocking accounts, manually locking and unlocking abnormal accounts, and deleting accounts that are no longer used.

Automatically Locking and Unlocking Accounts

- If the consecutive times that a user fails to enter the correct password reach the upper limit 10 during database connection, the system automatically locks the account for account security.
- An account that has been locked for one day is automatically unlocked.

Manually Locking and Unlocking Accounts

Once detecting that an account is stolen or the account is used to access the database without being authorized, administrators can manually lock the account.

The administrator can also manually unlock the account if the account becomes normal again.

For example, run the following command to manually lock and unlock the user `user_read`:

- To manually lock the account, run the following command:
`ALTER USER user_read ACCOUNT LOCK;`

If the following information is displayed, the user has been locked:

```
ALTER USER
```

- To manually unlock the account, run the following command:

```
ALTER USER user_read ACCOUNT UNLOCK;
```

If the following information is displayed, the user has been unlocked:

```
ALTER USER
```

Deleting Accounts that Are No Longer Used

An administrator can delete an account that is no longer used. This operation cannot be rolled back.

When an account to be deleted is in active state, it is deleted after the session is disconnected.

For example, run the following command to delete the account **user_read**:

```
DROP USER user_read CASCADE;
```

If the following information is displayed, the account has been deleted:

```
DROP USER
```

4.2.2 Setting the Validity Period of an Account

When creating a user, you need to specify the validity period of the user, including the start time and end time.

To enable a user not within the validity period to use its account, set a new validity period.

Procedure

- Step 1** Run the following command to create a user and specify the start time and end time of its validity period:

```
CREATE USER joe WITH PASSWORD 'Bigdata@123' VALID BEGIN '2015-10-10 08:00:00' VALID UNTIL '2016-10-10 08:00:00;
```

The user is created if the following information is displayed:

```
CREATE USER
```

- Step 2** If the user is not within the specified validity period, run the following command to set the start time and end time of a new validity period.

```
ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10 08:00:00;
```

The user is created if the following information is displayed:

```
ALTER USER
```

----End

NOTE

If **VALID BEGIN** is not specified in the **CREATE ROLE** or **ALTER ROLE** statement, the start time of the validity period is not limited. If **VALID UNTIL** is not specified, the end time of the validity period is not limited. If both of the parameters are not specified, the user is always valid.

4.2.3 Setting Password Security Policy

User passwords are stored in the system catalog **pg_authid**. To prevent password leakage, GaussDB(DWS) encrypts and stores the user passwords. In addition, the password security policy is restricted.

- Password complexity

The password complexity requirements are as follows:

- Minimum number of uppercase letters (A-Z) (**password_min_uppercase**)
- Minimum number of lowercase letters (a-z) (**password_min_lowercase**)
- Minimum number of digits (0-9) (**password_min_digits**)
- Minimum number of special characters (**password_min_special**) ([Table 4-3](#) lists special characters.)
- Minimum password length (**password_min_length**)
- Maximum password length (**password_max_length**)
- A password must contain at least three types of the preceding characters (uppercase letters, lowercase letters, digits, and special characters).
- A password must differ from the username or the username spelled backward.
- A new password must be different from the current password or the current password in reverse order.

- Password reuse

When a user changes the password, the user can reuse a password only if it has not been used for over 60 days.

- Password validity period

A validity period (90 days by default) is set for each database user password. If the password is about to expire (in seven days), the system displays a message reminding the user to change it upon login.

 NOTE

Considering the usage and service continuity of a database, the database still allows a user to log in after the password expires. A password change notification is displayed every time the user logs in to the database until the password is changed.

- Password change

- During database installation, an OS user with the same name as the initial user is created. The password of the OS user needs to be periodically changed for account security.

If the password of the user **user1** needs to be changed, run the following command:

```
passwd user1
```

Change the password as prompted.

- System administrators and common users need to periodically change their passwords to prevent the accounts from being stolen.

For example, to change the password of the user **user1**, connect to the database as the administrator and run the following command:

```
ALTER USER user1 IDENTIFIED BY "1234@abc" REPLACE "5678@def";
```

 NOTE

1234@abc and **5678@def** represent the new password and the original password of the user **user1**, respectively. If the new password does not have the required complexity, the change will not take effect.

- An administrator can change its own password and other accounts' passwords. If common users forget their passwords, they can ask administrators to change the passwords.

To change the password of user **joe**, run the following command:

```
ALTER USER joe IDENTIFIED BY "abc@1234";
```

 NOTE

- System administrators are not allowed to change passwords for each other.
 - A database administrator can change the password of a common user without being required to provide the common user's old password.
 - When a system administrator changes its own password, its original password is required.
 - Password verification
- Password verification is required when you set the user or role in the current session. If the entered password is inconsistent with the stored password of the user, an error is reported.

If user **joe** needs to be set, run the following command:

```
SET ROLE joe PASSWORD "abc@1234";
```

If the following information is displayed, the role setting has been modified:

```
SET ROLE
```

Table 4-3 Special characters

No.	Character	No.	Character	No.	Character	No.	Character
1	~	9	*	17		25	<
2	!	10	(	18	[	26	.
3	@	11	)	19	{	27	>
4	#	12	-	20	}	28	/
5	\$	13	_	21	]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	\	24	,	-	-

4.3 Viewing Audit Information

Prerequisites

- Auditing of required items has been enabled. For details about how to enable auditing, see [Configuring the Database Audit Log](#).
- The database is running properly and a series of addition, modification, deletion, and query operations have been executed in the database. Otherwise, no audit result is generated.
- Audit logs are separately recorded on the database nodes. If an LVS is used to manage the audit logs, you need to locate the specific node where the audit logs are recorded and connect the node to query the corresponding audit logs.

Context

- Only users with the **AUDITADMIN** permission can view audit records. For details about database users and how to create users, see [Users](#).
- The SQL function **pg_query_audit** provided by the database functions as the audit query command:

```
pg_query_audit(timestampz starttime,timestampz endtime,audit_log)
```

starttime and **endtime** indicate the start time and end time of the audit record, respectively. **audit_log** indicates the physical file path of the queried audit logs. If **audit_log** is not specified, the audit log information of the current instance is queried.

You can run the SQL function **pgxc_query_audit** to query audit logs of all CN nodes. The prototype is as follows:

```
pgxc_query_audit(timestampz starttime,timestampz endtime)
```

NOTE

The difference between the values of the **starttime** and **endtime** parameters indicates the query period, which can be any value ranging from 00:00:00 of the **starttime** parameter to 23:59:9 of the **endtime** parameter. Therefore, the **starttime** and **endtime** parameters must be properly set to ensure that the required audit information is displayed.

Procedure

Step 1 Use a SQL client to connect to the database.

Step 2 Run the following command to query the audit record:

```
SELECT * FROM pg_query_audit('2015-07-15 08:00:00','2015-07-15 09:47:33');
```

The query result is as follows.

time	type	result	username	database	client_conninfo	object_name
	detail_info		node_name	thread_id	local_port	remote_port
-----+-----+-----+-----+-----+-----+-----						
+-----+-----+-----+-----+-----+-----+-----						
+-----+-----+-----+-----+-----+-----+-----						
2015-07-15 08:03:55+08	login_success	ok	dbadmin	postgres	gs_clean@::1	postgres
login db(postgres) success,the current user is:dbadmin cn_5003						
139808902997776@490233835920483	9000	55805				

This audit record indicates that user **dbadmin** logged in to the **postgres** database at 08:03:55+08 on July 15, 2015. After the host specified by **log_hostname** is started and a client is connected to its IP address, the host name found by reverse DNS resolution is displayed following the at sign (@) in the value of **client_conninfo**.

Step 3 Run the following command to query the audit records of all CN nodes:

```
SELECT * FROM pgxc_query_audit('2019-01-10 17:00:00','2019-01-10 19:00:00') where type = 'login_success' and username = 'user1';
```

The query result is as follows:

time	type	result	username	database	client_conninfo	object_name
detail_info	node_name	thread_id				
local_port	remote_port					
-----+-----+-----+-----+-----+-----+-----						
+-----+-----+-----+-----+-----+-----+-----						
-----+-----						
2019-01-10 18:06:08+08	login_success	ok	user1	postgres	gresql@[local]	postgres
db(postgres) success,the current user is:user1	coordinator1	139965149210368@600429968516954				
17560	null					
2019-01-10 18:06:22+08	login_success	ok	user1	postgres	gresql@[local]	postgres
db(postgres) success,the current user is:user1	coordinator1	139965149210368@600429982697548				
17560	null					
2019-01-10 18:06:54+08	login_success	ok	user1	postgres	gresql@[local]	postgres
db(postgres) success,the current user is:user1	coordinator2	140677694355200@600430014804280				
17562	null					
(3 rows)						

The query result shows the successful login records of **user1** in to CN1 and CN2.

----End

5 Development and Design Proposal

5.1 Development and Design Proposal

This chapter describes the design specifications for database modeling and application development. Modeling compliant with these specifications fits the distributed processing architecture of GaussDB(DWS) and provides efficient SQL code.

The meaning of "Proposal" and "Notice" in this chapter is as follows:

- **Proposal:** Design rules. Services compliant with the rules can run efficiently, and those violating the rules may have low performance or logic errors.
- **Notice:** Details requiring attention during service development. This term identifies SQL behavior that complies with SQL standards but users may have misconceptions about, and default behavior that users may be unaware of in a program.

5.2 Database Object Naming Conventions

The name of a database object must contain 1 to 63 characters, start with a letter or underscore (_), and can contain letters, digits, underscores (_), dollar signs (\$), and number signs (#).

- [Proposal] Do not use reserved or non-reserved keywords to name database objects.

NOTE

To query the keywords of GaussDB(DWS), run **select * from pg_get_keywords()** or refer to section "Keyword."

- [Proposal] 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 problem location difficult.
- [Proposal] Use the same naming format for database objects.
 - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.

- A database object name consists of letters, digits, and underscores (_); and cannot start with a digit. You are advised to use multiple words separated with hyphens (-).
- You are advised to use intelligible names and common acronyms or abbreviations for database objects. Acronyms or abbreviations that are generally understood are recommended. For example, you can use English words 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.
- [Proposal] 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**.
 - Foreign tables are named in the format of **f_Suffix**.

5.3 Database Object Design

5.3.1 Database and Schema Design

In GaussDB(DWS), services can be isolated by databases and schemas. Databases share little resources and cannot directly access each other. Connections to and permissions on them are also isolated. Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be controlled using the **GRANT** and **REVOKE** syntax.

- You are advised to use schemas to isolate services for convenience and resource sharing.
- It is recommended that system administrators create schemas and databases and then assign required permissions to users.

Database Design

- [Proposal] Create databases as required by your business. Do not use the default **postgres** database of a cluster.
- [Proposal] Create a maximum of three user-defined databases in a cluster.
- [Proposal] To make your database compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.
- [Notice] When you create a database, exercise caution when you set **ENCODING** and **DBCMPATIBILITY** configuration items. In GaussDB(DWS), **DBCMPATIBILITY** can be set to **TD** or **Oracle** to be compatible with Teradata or Oracle syntax, respectively. Syntax behavior in the two modes is different. For details, see [Syntax Compatibility Differences Between Teradata and Oracle](#).
- [Notice] By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when using the deletion permission.

Schema Design

- [Notice] 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.
- [Notice] To let a user create an object in the schema, grant the create permission for the schema to the user.
- [Notice] By default, a schema owner has all permissions for all objects in the schema, including the deletion permission. Exercise caution when using the deletion permission.

5.3.2 Table Design

GaussDB(DWS) uses a distributed architecture. Data is distributed on DN. Comply with the following principles to properly design a table:

- [Notice] Evenly distribute data on each DN to prevent data skew. If most data is stored on several DNs, the effective capacity of a cluster decreases. Select a proper distribution column to avoid data skew.
- [Notice] 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.
- [Notice] Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.
- [Notice] Minimize random I/O. By clustering or local clustering, you can sequentially store hot data, converting random I/O to sequential I/O to reduce the cost of I/O scanning.
- [Notice] 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 to improve cluster performance and concurrency, you can minimize data shuffling by using proper association and grouping conditions.

Selecting a Storage Mode

[Proposal] Selecting a storage mode is the first step in defining a table. The storage mode mainly depends on the customer's service type. For details, see [Table 5-1](#).

Table 5-1 Table storage modes and scenarios

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

Storage Mode	Application Scenarios
Column storage	- Statistical analysis queries (requiring a large number of association and grouping operations) - Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Selecting a Distribution Mode

[Proposal] Comply with the following rules to distribute table data.

Table 5-2 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenarios
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 a cluster.	Dimension tables and fact tables containing a small amount of data

Selecting a Partitioning Mode

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

- [Proposal] Create partitions on columns that indicate certain ranges, such as dates and regions.
- [Proposal] A partition name should show the data characteristics of a partition. For example, its format can be Keyword+Range characteristics.
- [Proposal] Set the upper limit of a partition to **MAXVALUE** to prevent data overflow.

The example of a partitioned table definition 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)  
);
```

Selecting a Distribution Key

Selecting a distribution key is important for a hash table. An improper distribution key may cause data skew. As a result, the I/O load is heavy on several DN, affecting the overall query performance. After you select a distribution policy for a hash table, check for data skew to ensure that data is evenly distributed. Comply with the following rules to select a distribution key:

- [Proposal] Select a column containing discrete data as the distribution key, so that data can be evenly distributed on each DN. If 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.
- [Proposal] 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.
- [Proposal] 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.

5.3.3 Column Design

Selecting a Data Type

Comply with the following rules to improve query efficiency when you design columns:

- [Proposal] Use the most efficient data types allowed.
If all of the following number types provide the required service precision, they are recommended in descending order of priority: integer, floating point, and NUMERIC.
- [Proposal] In tables that are logically related, columns having the same meaning should use the same data type.
- [Proposal] 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 not advised to use CHAR(n), BPCHAR(n), NCHAR(n), or CHARACTER(n), unless you know that the string length is fixed.

For details about string types, see [Common String Types](#).

Common String Types

Every column requires a data type suitable for its data characteristics. The following table lists common string types in GaussDB(DWS).

Table 5-3 Common string types

Parameter	Description	Max. Storage Capacity
CHAR(<i>n</i>)	Fixed-length string, where <i>n</i> indicates the stored bytes. If the length of an input string is smaller than <i>n</i> , the string is automatically padded to <i>n</i> bytes using NULL characters.	10 MB
CHARACTER(<i>n</i>)	Fixed-length string, where <i>n</i> indicates the stored bytes. If the length of an input string is smaller than <i>n</i> , the string is automatically padded to <i>n</i> bytes using NULL characters.	10 MB
NCHAR(<i>n</i>)	Fixed-length string, where <i>n</i> indicates the stored bytes. If the length of an input string is smaller than <i>n</i> , the string is automatically padded to <i>n</i> bytes using NULL characters.	10 MB
BPCHAR(<i>n</i>)	Fixed-length string, where <i>n</i> indicates the stored bytes. If the length of an input string is smaller than <i>n</i> , the string is automatically padded to <i>n</i> bytes using NULL characters.	10 MB
VARCHAR(<i>n</i>)	Variable-length string, where <i>n</i> indicates the maximum number of bytes that can be stored.	10 MB
CHARACTER VARYING(<i>n</i>)	Variable-length string, where <i>n</i> indicates the maximum number of bytes that can be stored. This data type and VARCHAR(<i>n</i>) are different representations of the same data type.	10 MB
VARCHAR2(<i>n</i>)	Variable-length string, where <i>n</i> indicates the maximum number of bytes that can be stored. This data type is added to be compatible with the Oracle database, and its behavior is the same as that of VARCHAR(<i>n</i>).	10 MB
NVARCHAR2(<i>n</i>)	Variable-length string, where <i>n</i> indicates the maximum number of bytes that can be stored.	10 MB

Parameter	Description	Max. Storage Capacity
TEXT	Variable-length string. Its maximum length is 8203 bytes less than 1 GB.	8203 bytes less than 1 GB

5.3.4 Constraint Design

DEFAULT and NULL Constraints

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

Partial Cluster Key

A partial cluster key (PCK) is a local clustering technology used for column-store tables. After creating a PCK, you can quickly filter and scan fact tables using min or max sparse indexes in GaussDB(DWS). Comply with the following rules to create a PCK:

- [Notice] Only one PCK can be created in a table. A PCK can contain multiple columns, preferably no more than two columns.
- [Proposal] Create a PCK on simple expression filter conditions in a query. Such filter conditions are usually in the form of **col op const**, where **col** specifies a column name, **op** specifies an operator (such as =, >, >=, <=, and <), and **const** specifies a constant.
- [Proposal] If the preceding conditions are met, create a PCK on the column having the least distinct values.

Unique Constraint

- [Notice] Unique constraints can be used in row-store tables but not in column-store tables.
- [Proposal] The constraint name should indicate that it is a unique constraint, for example, **UNI***Included columns*.

Primary Key Constraint

- [Notice] Primary key constraints can be used in row-store tables but not in column-store tables.
- [Proposal] The constraint name should indicate that it is a primary key constraint, for example, **PK***Included columns*.

Check Constraint

- [Notice] Check constraints can be used in row-store tables but not in column-store tables.
- [Proposal] The constraint name should indicate that it is a check constraint, for example, **CK***Included columns*.

5.3.5 View and Joined Table Design

View Design

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

Joined Table Design

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

5.4 JDBC Configuration

Currently, third-party tools are connected to GaussDB(DWS) through JDBC. This section describes the precautions for configuring the tools.

Connection Parameters

- [Notice] When a third-party tool connects to GaussDB(DWS) through JDBC, JDBC sends a connection request to GaussDB(DWS). 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", "2" },  
  { "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.

- [Proposal] When connecting to the database through JDBC, ensure that the following two time zones are the same:
 - Time zone of the host where the JDBC client is located
 - Time zone of the host where the GaussDB(DWS) server is located

fetchsize

[Notice] To use **fetchsize** in applications, disable the **autocommit** switch. Enabling the **autocommit** switch makes the **fetchsize** configuration invalid.

autocommit

[Proposal] It is recommended that you enable the **autocommit** switch in the code for connecting to GaussDB(DWS) by the JDBC. If **autocommit** needs to be disabled to improve performance or for other purposes, applications need to ensure their transactions are committed. For example, explicitly commit transactions after specifying service SQL statements. Particularly, ensure that all transactions are committed before the client exits.

Connection Releasing

[Proposal] You are advised to use connection pools to limit the number of connections from applications. Do not connect to a database every time you run an SQL statement.

[Proposal] After an application completes its tasks, disconnect its connection to GaussDB(DWS) to release occupied resources. You are advised to set the session timeout interval in the task.

[Proposal] 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, before you return the connection to the connection pool, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status.
- If a temporary table is used, delete it before you return the connection to the connection pool.

CopyManager

[Proposal] 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 interface driven by the GaussDB(DWS) JDBC to import data in batches during application development.

5.5 SQL Compilation

DDL

- [Proposal] In GaussDB(DWS), you are advised to execute DDL operations, such as creating table or making comments, separately from batch processing jobs to avoid performance deterioration caused by many concurrent transactions.
- [Proposal] Execute data truncation after unlogged tables are used because GaussDB(DWS) cannot ensure the security of unlogged tables in abnormal scenarios.

- [Proposal] Suggestions on the storage mode of temporary and unlogged tables are the same as those on base tables. Create temporary tables in the same storage mode as the base tables to avoid high computing costs caused by hybrid row and column correlation.
- [Proposal] 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.
- [Proposal] 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 Uninstalling

- [Proposal] Provide the inserted column list in the insert statement. Example:
`INSERT INTO task(name,id,comment) VALUES ('task1','100','100th task');`
- [Proposal] 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.
- [Proposal] 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

- [Proposal] Perform type coercion to convert data types. If you perform implicit conversion, the result may differ from expected.
- [Proposal] During data query, explicitly specify the data type for constants, and do not attempt to perform any implicit data type conversion.
- [Notice] In Oracle compatibility mode, null strings will be automatically converted to NULL during data import. If a null string needs to be reserved, you need to create a database that is compatible with Teradata.

Query Operation

- [Proposal] 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.
- [Proposal] Perform DDL and DML operations encapsulated in transactions. Operations like table truncation, update, deletion, and dropping, cannot be rolled back once committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- [Proposal] During query compilation, you are advised to list all columns to be queried and avoid using *. Doing so reduces output lines, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- [Proposal] During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- [Proposal] The cost of joining more than three tables or views, especially full joins, is difficult to be estimated. You are advised to use the **WITH TABLE AS** statement to create interim tables to improve the readability of SQL statements.

- [Proposal] Do not use Cartesian products or full joins. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.
- [Notice] 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**.
- [Notice] 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.
- [Notice] 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.
- [Notice] To count multiple columns using count(), column names must be enclosed with 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.
- [Notice] Null records are not counted when count(distinct col) is used to calculate the number of non-null columns that are not repeated.
- [Notice] 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.
- [Notice] When constants are used to filter data, the system searches for functions used for calculating these two data types based on the data types of the constants and matched columns. If no function is found, the system converts the data type implicitly. Then, the system searches for a function used for calculating the converted data type.

```
SELECT * FROM test WHERE timestamp_col = 20000101;
```

In the preceding example, if **timestamp_col** is the timestamp type, the system first searches for the function that supports the "equal" operation of the timestamp and int types (constant numbers are considered as the int type). If no such function is found, the **timestamp_col** data and constant numbers are implicitly converted into the text type for calculation.

- [Proposal] 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 associations based on the service logic.

- [Proposal] In **WHERE** clauses, the filtering conditions should be sorted. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.
- [Proposal] Filtering conditions in **WHERE** clauses should comply with unilateral rules. That is, when the column name is placed on one side of a comparison operator, the optimizer automatically performs pruning optimization in some scenarios. Filtering conditions in a **WHERE** clause will be displayed in **col op expression** format, 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;
```

- [Proposal] Do not perform unnecessary sorting operations. Sorting 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, data in GaussDB(DWS) is sorted by ASC & NULL LAST.
- [Proposal] When the **ORDER BY** clause is used for sorting, specify sorting modes (ASC or DESC), and use NULL FIRST or NULL LAST for NULL record sorting.
- [proposal] 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.
- [Proposal] If the service logic is accurate, you are advised to use **UNION ALL** instead of **UNION**.
- [Proposal] If a filtering 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. 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)
```

- [Proposal] If an **in(val1, val2, va...)** expression contains a large number of columns, you are advised to replace it with the **in (values (va11), (val2), (val3...))** statement. The optimizer will automatically convert the **IN** constraint into a non-correlated subquery to improve the query performance.
- [Proposal] Replace **(not) in** with **(not) exist** when associated 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 T1,T2 WHERE T1.C1=T2.C2);
```

NOTE

- If the value of the T1.C1 column will possibly be NULL, the preceding rewriting cannot be performed.
- If T1.C1 is the output of a subquery, check whether the output is NOT NULL based on the service logic.
- [Proposal] 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 it and commit transaction once the query is finished.

6 Tutorial: Development Using JDBC or ODBC

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

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

6.2 Downloading Drivers

For details, see [Downloading the JDBC or ODBC Driver](#)

6.3 Using JDBC for Development

6.3.1 JDBC Development Process

For details, see [Using a JDBC Driver to Connect to the Database](#).

6.3.2 Example: Common Operations

Example 1

This example illustrates how to develop applications based on the GaussDB(DWS) JDBC interface.

```
//DBtest.java
//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the class name org.postgresql with
com.huawei.gauss200.jdbc and replace the URL prefix jdbc:postgresql with jdbc:gaussdb.
// This example illustrates the main processes of JDBC-based development, covering database connection
creation, table creation, and data insertion.

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.CallableStatement;

public class DBTest {

    //Establish a connection to the database.
    public static Connection GetConnection(String username, String passwd) {
        String driver = "org.postgresql.Driver";
        String sourceURL = "jdbc:postgresql://localhost:8000/postgres";
        Connection conn = null;
        try {
            //Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            //Establish a connection to the database.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    };

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

            //Run an ordinary 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 the preprocessing 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();
}
//Run batch processing.
pst.executeBatch();
pst.close();
} catch (SQLException e) {
    if (pst != null) {
        try {
            pst.close();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
    e.printStackTrace();
}
}

//Run the precompilation 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 {
        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 as an integer.
        cstmt.execute();
        int out = cstmt.getInt(4); //Obtain the out parameter value.
        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) {
    //Establish a connection to the database.
    Connection conn = GetConnection("tester", "Password1234");

    //Create a table.
    CreateTable(conn);

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

    //Run the precompilation 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, you need to disable the autocommit function.

```
// Disable the autocommit function.
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.");
}
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.");
}
rs.close();

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

6.3.3 Example: Retrying SQL Queries for Applications

If the primary DN is faulty and cannot be restored within 40s, its standby is automatically promoted to primary to ensure the normal running of the cluster. Jobs running during the failover will fail and those started after the failover will not be affected. To protect upper-layer services from being affected by the failover, refer to the following example to construct a SQL retry mechanism at the service layer.

//**gsjdbc4.jar** is used as an example. If **gsjdbc200.jar** is used, replace the class name **org.postgresql** with **com.huawei.gauss200.jdbc** and replace the URL prefix **jdbc:postgresql** with **jdbc:gaussdb**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 *
 */

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }

    public void run() {
        System.out.println("exit handle");
        try {
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}

public class SQLRetry {
    //Establish a connection to the database.
    public static Connection GetConnection(String username, String passwd) {
        String driver = "org.postgresql.Driver";
        String sourceURL = "jdbc:postgresql://10.131.72.136:8000/postgres";
        Connection conn = null;
        try {
            //Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            //Establish a connection to the database.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }
}
```

```
//Run an ordinary SQL statement. Create a jdbc_test1 table.
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();

        // add ctrl+c handler
        Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));

        // Run an ordinary 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 the preprocessing 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 the precompilation 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;
}

//Run a query statement and retry upon a failure. The number of retry times can be configured.
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 {
    //Establish a connection to the database.
    Connection conn = GetConnection("testuser", "test@123");

    //Create a table.
    CreateTable(conn);

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

    //Run the precompilation statement to update data.
    ExecPreparedSQL(conn);

    //Disconnect from the database.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



```
}  
  
}
```

6.3.4 Example: Importing and Exporting Data Through Local Files

When the JAVA language is used for secondary development based on GaussDB(DWS), you can use the CopyManager interface 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(DWS) JDBC driver before running it.

//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the class name **org.postgresql** with **com.huawei.gauss200.jdbc** and replace the URL prefix **jdbc:postgresql** with **jdbc:gaussdb**.

```
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://10.180.155.74:8000/postgres"); // URL of the database  
        String username = new String("jack"); //User name  
        String password = new String("Gauss@123"); //Password  
        String tablename = new String("migration_table"); //Define table information.  
        String tablename1 = new String("migration_table_1"); //Define 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) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            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) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            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) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

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();
            }
        }
    }
}

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();
            }
        }
    }
}
}
```

6.3.5 Example: Migrating Data from MySQL to GaussDB(DWS)

The following example shows how to use CopyManager to migrate data from the MySQL database to GaussDB(DWS).

```
//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the class name org.postgresql with com.huawei.gauss200.jdbc and replace the URL prefix jdbc:postgresql with jdbc:gaussdb.
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://10.180.155.74:8000/postgres"); //URL of the database
        String user = new String("jack"); //mppdb user name
        String pass = new String("Gauss@123"); //mppdb password
        String tablename = new String("migration_table"); //Define table information.
        String delimiter = new String("|"); //Define a delimiter.
        String encoding = new String("UTF8"); //Define a character set.
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer(); //Define the 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
            //newline character to form 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 table information.
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + "
ENCODING " + "'" + encoding + "'";

                //Submit 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 from the source database.
        //*****
        private static ResultSet getDataSet() {
            ResultSet rs = null;
            try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                Connection conn = DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "Gauss@123");
```

```
Statement stmt = conn.createStatement();
rs = stmt.executeQuery("select * from migration_table");
} catch (SQLException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
return rs;
}
```

6.4 JDBC Interface Reference

JDBC interface is a set of API methods for users. This section describes some common interfaces. For other interfaces, see information in JDK1.6 (software package) and JDBC4.0.

6.4.1 java.sql.Connection

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

Table 6-1 Support status for java.sql.Connection

Method Name	Return Type	Support JDBC 4
close()	void	Yes
commit()	void	Yes
createStatement()	Statement	Yes
getAutoCommit()	Boolean	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getTransactionIsolation()	int	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
prepareStatement(String sql)	PreparedStatement	Yes
rollback()	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes

NOTICE

The AutoCommit mode is used by default within the interface. If you disable it 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.

6.4.2 java.sql.CallableStatement

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

Table 6-2 Support status for java.sql.CallableStatement

Method Name	Return Type	Support JDBC 4
registerOutParameter(int parameterIndex, int type)	void	Yes
wasNull()	Boolean	Yes
getString(int parameterIndex)	String	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getShort(int parameterIndex)	short	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getFloat(int parameterIndex)	float	Yes
getDouble(int parameterIndex)	double	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBytes(int parameterIndex)	byte[]	Yes
getDate(int parameterIndex)	Date	Yes
getTime(int parameterIndex)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	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`.

6.4.3 java.sql.DatabaseMetaData

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

Table 6-3 Support status for `java.sql.DatabaseMetaData`

Method Name	Return Type	Support JDBC 4
<code>getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)</code>	ResultSet	Yes
<code>getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)</code>	ResultSet	Yes
<code>getTableTypes()</code>	ResultSet	Yes
<code>getUserName()</code>	String	Yes
<code>isReadOnly()</code>	boolean	Yes
<code>nullsAreSortedHigh()</code>	boolean	Yes
<code>nullsAreSortedLow()</code>	boolean	Yes
<code>nullsAreSortedAtStart()</code>	boolean	Yes
<code>nullsAreSortedAtEnd()</code>	boolean	Yes
<code>getDatabaseProductName()</code>	String	Yes
<code>getDatabaseProductVersion()</code>	String	Yes
<code>getDriverName()</code>	String	Yes
<code>getDriverVersion()</code>	String	Yes
<code>getDriverMajorVersion()</code>	int	Yes
<code>getDriverMinorVersion()</code>	int	Yes

Method Name	Return Type	Support JDBC 4
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNotNull()	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

Method Name	Return Type	Support JDBC 4
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes
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

6.4.4 java.sql.Driver

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

Table 6-4 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
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes

6.4.5 java.sql.PreparedStatement

This section describes **java.sql.PreparedStatement**, the interface for preparing statements.

Table 6-5 Support status for java.sql.PreparedStatement

Method Name	Return Type	Support JDBC 4
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
executeUpdate()	int	Yes
getMetaData()	ResultSetMetaData	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
setDouble(int parameterIndex, double x)	void	Yes

Method Name	Return Type	Support JDBC 4
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setNString(int parameterIndex, String value)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes
clearBatch()	void	Yes

 NOTE

- Execute addBatch() and execute() only after running clearBatch().
- Batch is not cleared by calling executeBatch(). Clear batch by explicitly calling clearBatch().
- After bounded variables of a batch are added, if you want to reuse these values (add a batch again), set*() is not necessary.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.

6.4.6 java.sql.ResultSet

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

Table 6-6 Support status for java.sql.ResultSet

Method Name	Return Type	Support JDBC 4
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes

Method Name	Return Type	Support JDBC 4
getBigDecimal(String columnLabel)	BigDecimal	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
getBytes(String columnLabel)	byte[]	Yes
getDate(int columnIndex)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	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
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes

Method Name	Return Type	Support JDBC 4
getTime(int columnIndex)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isFirst()	Boolean	Yes
next()	Boolean	Yes

 NOTE

- One Statement cannot have multiple open ResultSets.
- The cursor that is used for traversing the ResultSet cannot be open after committed.

6.4.7 java.sql.ResultSetMetaData

This section describes **java.sql.ResultSetMetaData**, which provides details about ResultSet object information.

Table 6-7 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	Support JDBC 4
getColumnCount()	int	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

6.4.8 java.sql.Statement

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

Table 6-8 Support status for java.sql.Statement

Method Name	Return Type	Support JDBC 4
close()	void	Yes
execute(String sql)	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
getConnection()	Connection	Yes
getResultSet()	ResultSet	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
isClosed()	Boolean	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes

 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 transaction. If **setFetchSize** is set, set **setAutoCommit(false)** and commit transactions on the connection to flush service data to a database.

6.4.9 javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the interface for data source connection pools.

Table 6-9 Support status for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	Support JDBC 4
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes

Method Name	Return Type	Support JDBC 4
getPooledConnection(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

6.4.10 javax.sql.DataSource

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

Table 6-10 Support status for javax.sql.DataSource

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

6.4.11 javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection interface created by a connection pool.

Table 6-11 Support status for javax.sql.PooledConnection

Method Name	Return Type	Support JDBC 4
addConnectionEventListener(ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes

Method Name	Return Type	Support JDBC 4
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

6.4.12 javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 6-12 Support status for javax.naming.Context

Method Name	Return Type	Support JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

6.4.13 javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 6-13 Support status for javax.naming.spi.InitialContextFactory

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

6.4.14 CopyManager

CopyManager is an API interface class provided by the JDBC driver in GaussDB(DWS). It is used to import data to GaussDB(DWS) in batches.

Inheritance Relationship of CopyManager

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

```
public class CopyManager  
extends Object
```

Construction Method

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

Basic Methods

Table 6-14 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 to tables in the database from InputStream.	SQLException,IOException

Return Value	Method	Description	throws
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 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

6.5 Using ODBC for Development

6.5.1 ODBC Development Process

For details, see [Using an ODBC Driver to Connect to the Database](#)

6.5.2 Examples

Code for Common Functions

```
// The following example shows how to obtain data from GaussDB(DWS) through the ODBC interface.  
// 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;
SQLLEN V_StrLen_or_IndPtr;
int main(int argc,char *argv[])
{
    // 1. Apply for 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. Apply for 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);
    }
    // 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. Set them as needed.
    // If the username and password have been set in the odbc.ini file, you do not need to set userName or
    // password here, retaining "" for them. However, you are not advised 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. Apply for a statement handle
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. Executes an SQL statement directly
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
    VARCHAR(32));",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,'li')",SQL_NTS);
    // 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. Execute the ready statement
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
    // 12. Obtain the attributes of a certain column in the result set
    SQLColAttribute(V_OD_hstmt,
    1,SQL_DESC_TYPE_NAME,typename,sizeof(typename),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_StrLen_or_IndPtr);
    // 14. Collect data using SQLFetch
    V_OD_erg=SQLFetch(V_OD_hstmt);
    // 15. Obtain and return data using 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 from the data source and release handles
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
return(0);
}
```

Code for Batch Processing

```
/* *****
* Set UseBatchProtocol to 1 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.
* *****/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

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);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
                hstmt, SQL_HANDLE_STMT);

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);
}

int main ()
{
    SQLHENV henv = SQL_NULL_HENV;
    SQLHDBC hdbc = SQL_NULL_HDBC;
    int batchCount = 1000;
    SQLLEN rowsCount = 0;
    int ignoreCount = 0;

    SQLRETURN retcode;
    SQLCHAR dsn[1024] = {"\0"};
    SQLCHAR loginfo[2048];

    // Interactively obtain data source names.
    getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
    // Interactively obtain the amount of data to be batch processed.
```

```
getInt("batchCount", &batchCount, 'N', 1);
do
{
// Interactively obtain the amount of batch processing data that is not inserted into the database.
getInt("ignoreCount", &ignoreCount, 'N', 1);
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);
CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_ENV)",
henv, SQL_HANDLE_ENV);

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

// Allocate Connection
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
henv, SQL_HANDLE_DBC);

// Set Login Timeout
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
hdbc, SQL_HANDLE_DBC);

// Set Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)(1), 0);
CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_ATTR_AUTOCOMMIT)",
hdbc, SQL_HANDLE_DBC);

// Connect to DSN
sprintf(logininfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
(SQLCHAR*) NULL, 0, NULL, 0);
CHECK_ERROR(retcode, logininfo, hdbc, SQL_HANDLE_DBC);

// 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 hstmtinsrt = SQL_NULL_HSTMT;
int i;
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);
// 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.
That is, 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 (int 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);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
        hstmtinesrt, SQL_HANDLE_STMT);

    // 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);
    CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));
    CHECK_ERROR(retcode, "SQLSetStmtAttr", hstmtinesrt, SQL_HANDLE_STMT);

    retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
    CHECK_ERROR(retcode, "SQLBindParameter for id", hstmtinesrt, SQL_HANDLE_STMT);

    retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);
    CHECK_ERROR(retcode, "SQLBindParameter for cols", hstmtinesrt, SQL_HANDLE_STMT);

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));
    CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAMS_PROCESSED_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);
    CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_STATUS_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);
    CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_OPERATION_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

    retcode = SQLRowCount(hstmtinesrt, &rowsCount);
    CHECK_ERROR(retcode, "SQLRowCount execution", hstmtinesrt, SQL_HANDLE_STMT);

    if (rowsCount != (batchCount - ignoreCount))
    {
```

```
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);
        CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
    }
    else
    {
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);
        CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
    }

    if (rowsCount != process)
    {
        sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
        CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
    }
    else
    {
        sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
        CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
    }

    for (int 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]);
                CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
        }
        else if (statusptr[i] != SQL_PARAM_SUCCESS)
        {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
        }
    }

    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
    sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");
    CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);
}

exit:
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;
}
```

6.6 ODBC Interfaces

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)).

6.6.1 SQLAllocEnv

In ODBC 3.x, **SQLAllocEnv** (an ODBC 2.x function) was deprecated and replaced with **SQLAllocHandle**. For details, see [SQLAllocHandle](#).

6.6.2 SQLAllocConnect

In ODBC 3.x, **SQLAllocConnect** (an ODBC 2.x function) was deprecated and replaced with **SQLAllocHandle**. For details, see [SQLAllocHandle](#).

6.6.3 SQLAllocHandle

Function

SQLAllocHandle allocates environment, connection, or statement handles. This function is a generic function for allocating handles that replaces the deprecated ODBC 2.x functions **SQLAllocEnv**, **SQLAllocConnect**, and **SQLAllocStmt**.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT HandleType,  
                          SQLHANDLE InputHandle,  
                          SQLHANDLE *OutputHandlePtr);
```

Parameter

Table 6-15 SQLAllocHandle parameters

Keyword	Description
HandleType	The 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 (description handle) The handle application sequence is: SQL_HANDLE_ENV > SQL_HANDLE_DBC > SQL_HANDLE_STMT . 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 is SQL_NULL_HANDLE. • If HandleType is SQL_HANDLE_DBC, this must be an environment handle. • If HandleType is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.

Keyword	Description
OutputHandlePtr	Output parameter: Pointer to a buffer in which to return the handle to the newly allocated data structure.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

When allocating a non-environment handle, if **SQLAllocHandle** returns **SQL_ERROR**, it sets **OutputHandlePtr** to **SQL_NULL_HENV**, **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **InputHandle**, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.4 SQLAllocStmt

In ODBC 3.x, **SQLAllocStmt** was deprecated and replaced with **SQLAllocHandle**. For details, see [SQLAllocHandle](#).

6.6.5 SQLBindCol

Function

SQLBindCol is used to associate (bind) columns in a result set to an application data buffer.

Prototype

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


Parameter

Table 6-16 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 at 1.
TargetType	The C data type in the buffer.
TargetValuePtr	Output parameter: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If TargetValuePtr is null, StrLen_or_IndPtr is a valid value.
BufferLength	Size of the TargetValuePtr buffer in bytes available to store the column data.
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 Values

- SQL_SUCCESS indicates that the call succeeded.
- SQL_SUCCESS_WITH_INFO indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.6 SQLBindParameter

Function

SQLBindParameter is used to associate (bind) parameter markers in an SQL statement to a buffer.

Prototype

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

Parameter

Table 6-17 SQLBindParameter

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting at 1 and increasing in an 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	Digital number 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 StrLen_or_IndPtr is null, no length or indicator is used.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.7 SQLColAttribute

Function

SQLColAttribute 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 6-18 SQLColAttribute parameter

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting at 1 and increasing in an ascending order.
FieldIdentifier	Field identifier of ColumnNumber in IRD.
CharacterAttributePtr	Output parameter: pointer to the buffer that returns FieldIdentifier field value.

Keyword	Description
BufferLength	• FieldIdentifier 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 Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLColAttribute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.8 SQLConnect

Function

SQLConnect establishes a connection between a driver and a data source. After the connection, 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 6-19 SQLConnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle .
ServerName	Name of the data source to connect to.
NameLength1	Length of ServerName .
UserName	User name of the database in the data source.
NameLength2	Length of UserName .
Authentication	User password of the database in the data source.
NameLength3	Length of Authentication .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLConnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.9 SQLDisconnect

Function

SQLDisconnect closes the connection associated with the database connection handle.

Prototype

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

Parameter

Table 6-20 SQLDisconnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLDisconnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.10 SQLExecDirect

Function

SQLExecDirect executes a prepared SQL statement specified in this parameter. This is the fastest execution method for executing only one SQL statement at a time.

Prototype

```
SQLRETURN SQLExecDirect(SQLHSTMT StatementHandle,  
                        SQLCHAR *StatementText,  
                        SQLINTEGER TextLength);
```

Parameter

Table 6-21 SQLExecDirect parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle .
StatementText	SQL statement to be executed. One SQL statement can be executed at a time.
TextLength	Length of StatementText .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_NEED_DATA** indicates insufficient parameters provided before executing the SQL statement.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.
- **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 then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.11 SQLExecute

Function

The **SQLExecute** function executes a prepared SQL statement using **SQLPrepare**. The statement is executed using the current value of any application variables that were bound to parameter markers by **SQLBindParameter**.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT StatementHandle);
```

Parameter

Table 6-22 SQLExecute parameters

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_NEED_DATA** indicates insufficient parameters provided before executing the SQL statement.
- **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. Values returned by other APIs are similar to the preceding values.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLExecute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.12 SQLFetch

Function

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

Prototype

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


Parameter

Table 6-23 SQLFetch parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle .

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.13 SQLFreeStmt

In ODBC 3.x, **SQLFreeStmt** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see [SQLFreeHandle](#).

6.6.14 SQLFreeConnect

In ODBC 3.x, **SQLFreeConnect** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see [SQLFreeHandle](#).

6.6.15 SQLFreeHandle

Function

SQLFreeHandle 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 6-24 SQLFreeHandle parameters

Keyword	Description
HandleType	The 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	The name of the handle to be freed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

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

Examples

See [Examples](#).

6.6.16 SQLFreeEnv

In ODBC 3.x, **SQLFreeEnv** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see [SQLFreeHandle](#).

6.6.17 SQLPrepare

Function

SQLPrepare prepares an SQL statement to be executed.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT StatementHandle,  
                     SQLCHAR *StatementText,  
                     SQLINTEGER TextLength);
```

Parameter

Table 6-25 SQLPrepare parameters

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

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLPrepare** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.18 SQLGetData

Function

SQLGetData retrieves data for a single column in the current row of 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  
SQLLEN    BufferLength,  
          *StrLen_or_IndPtr);
```

Parameter

Table 6-26 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 Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.19 SQLGetDiagRec

Function

SQLGetDiagRec returns the current values of multiple fields of 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 6-27 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 shared or non-shared environment handle.
RecNumber	Indicates the 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.
BufferLength	Length of MessageText.

Keyword	Description
TextLengthPtr	Output parameter: pointer to the buffer, the total number of bytes in the returned MessageText . If the number of bytes available to return is greater than BufferLength , then the diagnostics information text in MessageText is truncated to BufferLength minus the length of the null termination character.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

SQLGetDiagRec does not release diagnostic records for itself. It uses the following returned values to report execution results:

- **SQL_SUCCESS:** The function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO:** The ***MessageText** buffer is too small to hold the requested diagnostic message. No diagnostic records are generated.
- **SQL_INVALID_HANDLE:** The handle indicated by **HandType** and **Handle** is not a valid handle.
- **SQL_ERROR:** **RecNumber** is smaller than or equal to zero, or **BufferLength** is smaller than zero.

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

Table 6-28 SQLSTATE values

SQLSTATE	Error	Description
HY000	General error	An error occurred for which there is no specific SQLSTATE.
HY001	Memory allocation error	The driver is unable to allocate memory required to support execution or completion of the function.

SQLSATATE	Error	Description
HY008	Operation canceled	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timed out	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver	The called function is not supported by the StatementHandle driver.

Examples

See [Examples](#).

6.6.20 SQLSetConnectAttr

Function

SQLSetConnectAttr sets connection attributes.

Prototype

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

Parameter

Table 6-29 SQLSetConnectAttr parameters

Keyword	Description
StatementHandle	Connection handle.
Attribute	Attribute to set.

Keyword	Description
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 ValuePtr parameter is driver-specific value, it may be signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLSetConnectAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.21 SQLSetEnvAttr

Function

SQLSetEnvAttr sets environment attributes.

Prototype

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


Parameters

Table 6-30 SQLSetEnvAttr parameters

Keyword	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. Its 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, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLSetEnvAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

6.6.22 SQLSetStmtAttr

Function

SQLSetStmtAttr sets attributes related to a statement.

Prototype

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

Parameter

Table 6-31 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, and a driver-specified value. If ValuePtr parameter is driver-specific value, it may be signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates 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. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLSetStmtAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See [Examples](#).

7 Data Import

7.1 Import Modes

GaussDB(DWS) provides flexible methods for importing data. You can import data from different sources to GaussDB(DWS). For details, see [Figure 7-1](#). Each method has its own features, as listed in [Table 7-1](#). You can select a method that meets your requirements. It is recommended that Cloud Data Migration (CDM), Data Ingestion Service (DIS), and Data Lake Factory (DLF) services be used to import and export data. CDM is used for batch data migration. DIS is used for stream data ingestion. DLF orchestrates and schedules the entire ETL process and provides a visualized development environment.

Figure 7-1 Import modes

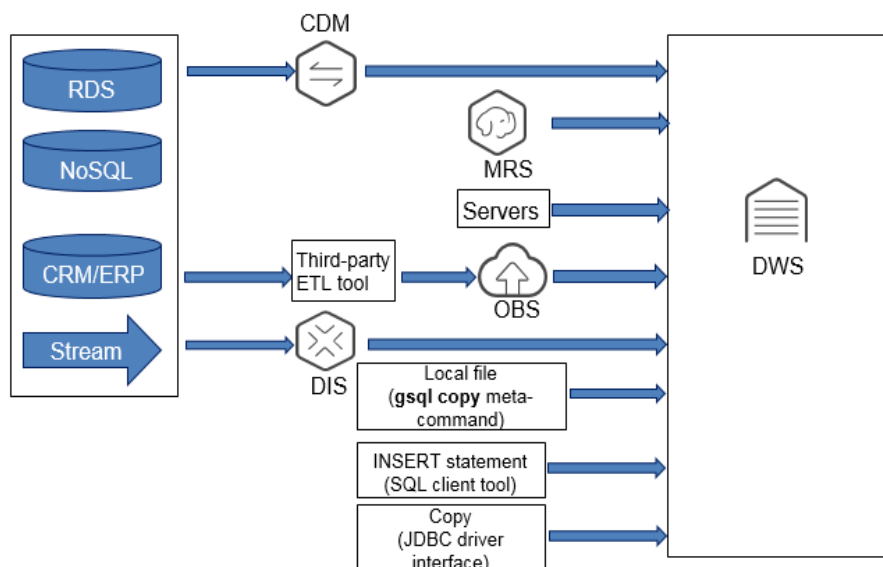


Table 7-1 Import modes

Import Mode	Data Source	Description	Advantage
Importing Data from OBS in Parallel	OBS	You can import data in TXT, CSV, ORC, or CARBONDATA format from OBS to GaussDB(DWS) for query, and can remotely read data from OBS. It is recommended for GaussDB(DWS).	This method features high performance and horizontal scaling.
Using GDS to Import Data from a Remote Server	Servers (remote servers)	Use the GDS tool provided by GaussDB(DWS) to import data from the remote server to GaussDB(DWS) in parallel. Multiple DNs are used for the import. This mode is efficient and suitable for importing a large amount of data to the database.	
Importing Data from MRS to a Cluster	MRS (HDFS)	Configure a GaussDB(DWS) cluster to connect to an MRS cluster. In GaussDB(DWS), read data from the HDFS of MRS.	This method features high performance and horizontal scaling.
Using CDM to Migrate Data to GaussDB(DWS)	Databases, NoSQL, file systems, and big data platforms	CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources. CDM migrates data to GaussDB(DWS) using the copy method or the GDS parallel import method.	Rich data sources and simple operations
Importing Stream Data from DIS to GaussDB(DWS)	Stream data	You can periodically import stream data from DIS to the database of a GaussDB(DWS) cluster. Data is temporarily stored on OBS before being imported to GaussDB(DWS) and will be deleted after it is dumped to GaussDB(DWS).	Quasi-real-time data import

Import Mode	Data Source	Description	Advantage
Third-party ETL tool	Databases, NoSQL, file systems, and big data platforms	For details, see the documents of the third-party ETL tool. GaussDB(DWS) provides the DSC tool to migrate Teradata/Oracle scripts to GaussDB(DWS). • Using the DSC Tool to Migrate SQL Scripts • Using Schema Convertor Tool to Migrate MySQL to DWS	Provides abundant data sources and powerful data conversion capabilities through OBS.
Running the INSERT Statement to Insert Data	-	If you use an SQL client tool or JDBC/ODBC driver to connect to the GaussDB(DWS) database, you can run the INSERT statement to insert one or more rows of data or to insert data from a specified table.	INSERT is the simplest way to write data. It is applicable to scenarios with small data volume and low concurrency.
Running the COPY FROM STDIN Statement to Import Data	Other files or databases	When you use Java to develop applications, the CopyManager interface of the JDBC driver is invoked to write data from files or other databases to GaussDB(DWS).	Data is directly written from other databases to GaussDB(DWS). Service data does not need to be stored in files.
Using a gsql Meta-Command \copy to Import Data	Local file	Different from the SQL COPY statement, the \copy command can read data from or write data into only local files on a gsql client.	This method is easy to operate and suitable for importing a small amount of data to the database.

7.2 Concurrently Importing Data from OBS

NOTICE

- Ensure no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the data warehouse cluster are in the same region.

7.2.1 About Parallel Data Import from OBS

Object Storage Service (OBS) is an object-based massive storage service provided by the public cloud, featuring data storage of high security, proven reliability, and cost-effectiveness. OBS provides large storage capacity for you to store files of any type.

GaussDB(DWS), a data warehouse service, uses OBS as a platform for converting cluster data and external data, satisfying the requirements for secure, reliable, and cost-effective storage.

You can import data in TXT, CSV, ORC, or CARBONDATA format from OBS to GaussDB(DWS) for query, and can remotely read data from OBS. You are advised to import frequently accessed hot data to GaussDB(DWS) to facilitate queries and store cold data to OBS for remote read to reduce cost.

Currently, the operations on ORC and CARBONDATA files are different from those on TXT and CSV files.

- Import or query data in TXT or CSV format
You do not need to customize external servers. Remotely reading data in TXT or CSV format is slow. [About Parallel Data Import from OBS](#) describes how to import data in TXT or CSV format from OBS to a cluster. To remotely read data, simply do **SELECT** to the foreign table created in [Creating an OBS Foreign Table](#).
- Import or query data in ORC format
You need to customize an external server. [OBS Query Overview](#) describes how to perform the operations on ORC data on OBS. Its subsection [Querying Data on OBS Through Foreign Tables](#) describes how to query data in ORC format through a foreign table and how to import and then query data.
- Import or query data in CARBONDATA format
You need to customize an external server. The import and query methods are the same as those of the ORC format.

Overview

During data migration and Extract-Transform-Load (ETL), a massive volume of data needs to be imported to GaussDB(DWS) in parallel. The common import mode is time-consuming. When you import data in parallel using OBS foreign tables, source data files to be imported are identified based on the import URL

and data formats specified in the tables. Data is imported in parallel through DNs to GaussDB(DWS), which improves the overall import performance.

Advantages:

- The CN only plans and delivers data import tasks, and the DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to import data.
- You can preprocess data before the import.
- Fault tolerance can be configured for data format errors during the data import. You can locate incorrect data based on displayed error information after the data is imported.

Disadvantage:

You need to create OBS foreign tables and store to-be-imported data on OBS.

Application Scenario:

A large volume of local data is imported concurrently on many DNs.

Related Concepts

- **Source data file:** a TEXT, CSV, ORC, or CARBONDATA file that stores data to be imported in parallel.
- **OBS:** a cloud storage service used to store unstructured data, such as documents, images, and videos. Data is imported in parallel from the OBS server to GaussDB(DWS).
- **Bucket:** a container storing objects on OBS.
 - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
 - In OBS, each bucket name must be unique and cannot be changed. A default ACL is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.
 - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object:** a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**. Generally, objects are managed as files. However, OBS has no file system-related concepts, such as files and folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key:** name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name+Object name*.

- **Metadata:** object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
 - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
 - User metadata contains object descriptions specified by users for uploading objects.
- **Data:** object content. OBS does not sense the content and regards it as stateless binary data.
- **Ordinary table:** A database table that stores data imported to data files in parallel. Ordinary tables are classified into row-store tables and column-store tables.
- **Foreign table:** A foreign table is used to identify data in a source data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a source data file.

How Data Is Imported

Figure 7-2 shows how data is imported from OBS. The CN plans and delivers data import tasks. It delivers tasks to each DN by file.

The allocation method is as follows:

In **Figure 7-2**, there are four DNs (DN0 to DN3) and OBS stores six files numbered from t1.data.0 to t1.data.5. The files are assigned as follows:

t1.data.0 -> DN0

t1.data.1 -> DN1

t1.data.2 -> DN2

t1.data.3 -> DN3

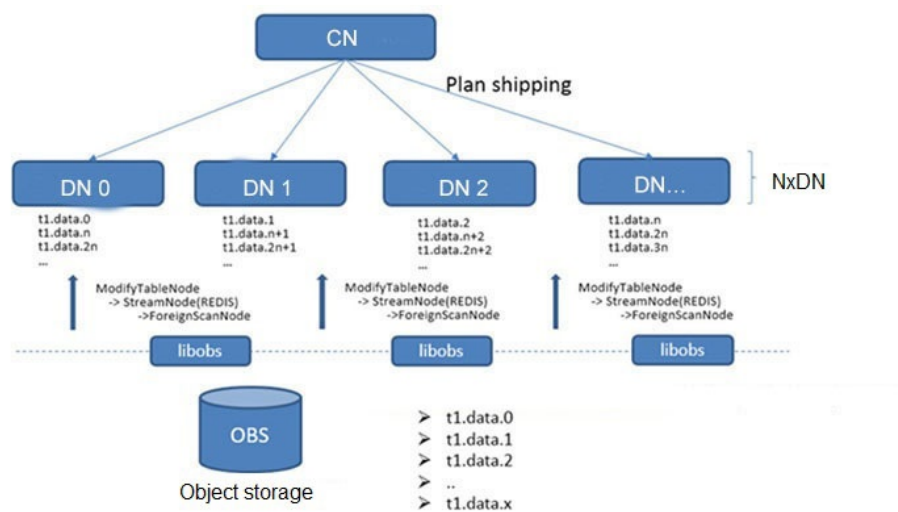
t1.data.4 -> DN0

t1.data.5 -> DN1

DN0 and DN1 each are assigned with two files. Other DNs each are assigned with one file.

The import performance is the best when each DN is assigned with one OBS file and all the files have the same size. To improve the performance of loading data from OBS, split the data file into multiple files as evenly as possible before storing it to OBS. The recommended number of split files is an integer multiple of the DN quantity.

Figure 7-2 Importing data using OBS foreign tables



Import Process

Figure 7-3 Parallel import procedure

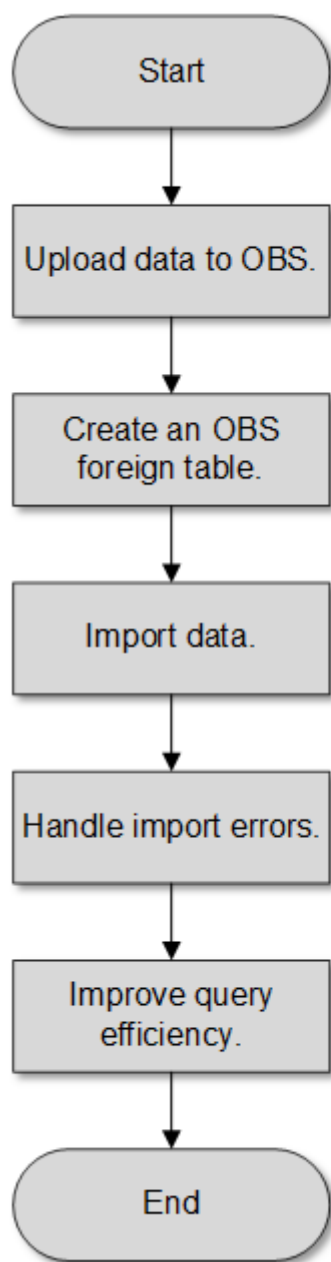


Table 7-2 Process description

Procedure	Description	Subtask
Upload data to OBS.	Plan the storage path on the OBS server and upload data files. For details, see Uploading Data to OBS .	-

Procedure	Description	Subtask
Create an OBS foreign table.	Create a foreign table to identify source data files on the OBS server. The OBS foreign table stores data source information, such as its bucket name, object name, file format, storage location, encoding, and delimiters. For details, see Creating an OBS Foreign Table .	-
Import data.	After creating the foreign table, run the INSERT statement to efficiently import data to the target tables. For details, see Importing Data .	-
Handle the table with import errors.	If errors occur during data import, handle them based on the displayed error information described in Handling Import Errors to ensure data integrity.	-
Improve query efficiency.	After data is imported, run the ANALYZE statement to generate table statistics. The ANALYZE statement stores the statistics in the PG_STATISTIC system catalog. The execution plan generator uses the statistics to generate the most efficient query execution plan. For details, see Analyzing a Table .	-

7.2.2 Creating Access Keys (AK and SK)

In this example, OBS data is imported to GaussDB(DWS) databases. When users who have registered with the public cloud use clients, call APIs, or use SDKs to access OBS, access keys (AK/SK) are required for user authentication. Therefore, if you want to connect to the GaussDB(DWS) database through a client or a JDBC/ODBC application to access OBS, obtain the access keys (AK and SK) first.

- Access Key ID (AK): indicates the ID of the access key, which is a unique identifier used in conjunction with a secret access key to sign requests cryptographically.
- Secret access key (SK): indicates key used together with its associated AK to cryptographically sign requests. The AK and SK are used together to identify a request sender to prevent the request from being modified.

Creating Access Keys (AK and SK)

Before creating the AK/SK pair, ensure that your account (used to log in to the management console) has passed real-name authentication.

To create an AK and an SK on the OBS console, perform the following steps:

Step 1 Log in to the GaussDB(DWS) management console.

Step 2 Move the pointer to the username in the upper right corner and choose **My Credentials**

Step 3 In the left navigation tree, click **Access Keys**.

If an access key already exists in the access key list, you can directly use it. However, you can view only **Access Key ID** in the access key list. You can download the key file containing the AK and SK only when adding an access key. If you do not have the key file, click **Create Access Key** to create one.

NOTE

- Each user can create a maximum of two valid access keys. If two access keys exist, delete one of them and create a new one. To delete an access key, you need to enter the current login password and email address or SMS verification code. Deletion is successful only after the verification is passed.
- To ensure account security, change your AK/SK pairs periodically and keep them safe.

Step 4 Click **Create Access Key**.

Step 5 In the **Create Access Key** dialog box that is displayed, enter the password and the verification code and click **OK**.

NOTE

- If you have not bound an email address or mobile phone number, enter only the password.
- If you have bound an email address and a mobile phone number, you can use either of them for verification.

Step 6 In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to default download path of your browser.

NOTE

- To prevent the access key from being leaked, keep it secure.
- If you click **Cancel** in the dialog box, the AKs will not be downloaded, and you cannot download them later. In this case, re-create an AK.

Step 7 Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

----End

Precautions

If you find that your AK/SK pair is abnormally used (for example, the AK/SK pair is lost or leaked) or will be no longer used, delete your AK/SK pair in the access key list or contact the administrator to reset your AK/SK pair.

When deleting the access keys, you need to enter the login password and either an email or mobile verification code.

NOTE

Deleted AK/SK pairs cannot be restored.

7.2.3 Uploading Data to OBS

Scenarios

Before importing data from OBS to a cluster, prepare source data files and upload these files to OBS. If the data files have been stored on OBS, you only need to complete [Step 2](#) to [Step 3](#) in [Uploading Data to OBS](#).

Preparing Data Files

Prepare source data files to be uploaded to OBS. GaussDB(DWS) supports only source data files in CSV, TEXT, ORC, or CARBONDATA format.

If user data cannot be saved in CSV format, store the data as any text file.

NOTE

According to [How Data Is Imported](#), when the data volume of each source data file is large, evenly split these files into multiple files before storing them to OBS. The excellent import performance is delivered when the number of files is an integer multiple of the DN quantity.

Uploading Data to OBS

Step 1 Upload data to OBS.

Store the source data files to be imported in the OBS bucket in advance.

1. Log in to the OBS management console.
Click **Service List** and choose **Object Storage Service** to open the OBS management console.
Alternatively, you can access <https://storage.huaweicloud.com> to log in to the OBS management console.
2. Create a bucket.
For details about how to create a bucket, see [Creating a Bucket](#) in the *Object Storage Service Console Operation Guide*.
For example, create two buckets named **mybucket** and **mybucket02**.
3. Create a folder.
For details, see [Creating a Folder](#) in the *Object Storage Service Console Operation Guide*.

For example:

- Create a folder named **input_data** in the **mybucket** OBS bucket.
- Create a folder named **input_data** in the **mybucket02** OBS bucket.

4. Upload the files.

For details, see [Uploading an Object](#) in the *Object Storage Service Console Operation Guide*.

For example:

- Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:
product_info.0
product_info.1
- Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:
product_info.2

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. The OBS path of the source data files is the value of the **location** parameter used for creating a foreign table.

The OBS folder path in the **location** parameter consists of **obs://**, a bucket name, and a file path. Example:

obs://<bucket_name>/<file_path>

For example, the OBS paths are as follows:

```
obs://mybucket/input_data/product_info.0
obs://mybucket/input_data/product_info.1
obs://mybucket02/input_data/product_info.2
```

Step 3 Grant the OBS bucket read permission for the user who will import data.

When importing data from OBS to a cluster, the user must have the read permission for the OBS buckets where the source data files are located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see [Configuring a Bucket ACL](#) in the *Object Storage Service Console Operation Guide*.

----End

7.2.4 Creating an OBS Foreign Table

Procedure

Step 1 Set **location** of the foreign table based on the path planned in [Uploading Data to OBS](#).

Step 2 Obtain access keys (AK and SK) for accessing OBS. To obtain access keys, log in to the management console, move the cursor over the username in the upper right corner, and click **My Credentials**. Then choose **Access Keys** in the navigation tree on the left. On the **Access Keys** page, you can view the existing AKs or click **Create Access Key** to create an AK/SK pair.

- Step 3** Set data format parameters for the foreign table based on the formats of data to be imported. You need to collect the following source data information:
- **format**: format of the source data file in the foreign table. OBS foreign tables support CSV and TEXT formats. The default value is **TEXT**.
 - **header**: Whether the data file contains a table header. Only CSV files can have headers.
 - **delimiter**: Delimiter specified to separate data fields in a file. If no delimiter is specified, the default one will be used.
 - For more parameters used for foreign tables, see data format parameters.

- Step 4** Plan the error tolerance of parallel import to specify how errors are handled during the import.
- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.
 - **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.
 - **per node reject limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import fails and an error message will be reported. This parameter is optional.
 - **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

The following describes the rules for converting an invalid character:

- **\0** is converted to a space.
- Other invalid characters are converted to question marks (?).
- If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to modify parameter settings that may cause import errors.
- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- For details about the parameters, see error tolerance parameters.

- Step 5** Create an OBS table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

The following describes the parameters required for creating a foreign table in the GaussDB(DWS) database:

- **Data format parameter access keys (AK and SK)**

- Set **access_key** to the AK you have obtained.
- Set **secret_access_key** to the SK you have obtained.

 NOTE

The values of **access_key** and **secret_access_key** are examples only.

- **Set data format parameters.**

- Set **format** to **CSV**.
- Set **encoding** to **UTF-8**.
- Set **encrypt** to **off**.
- Set **delimiter** to **,**.
- Retain the default value (double quotation marks) of **quote**.
- Set **null** (null value in a source data file) to a null string without quotation marks.
- Set **header** (whether the exported data file contains the header row) to the default value **false**. If the first row of the data file is not a header, retain the default value.

 NOTE

When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**.

- **Set fault-tolerant parameters for data import.**

- Set **PER NODE REJECT LIMIT** to **'unlimited'**, indicating that all data format errors detected during data import are allowed.
- Set **LOG INTO** to **product_info_err**, indicating that the data format errors detected during data import are recorded in the **product_info_err** table.
- If **fill_missing_fields** is set to **true** and the last column of a data row in a source data file is lost, the column is replaced with **NULL** and no error message will be generated.
- If **ignore_extra_data** is set to **true** and the number of columns of the source data file is greater than that defined for the foreign table, the extra columns at the end of the row are ignored and no error message will be generated.

Based on the preceding settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE product_info_ext;  
  
CREATE FOREIGN TABLE product_info_ext  
(  
    product_price          integer      not null,  
    product_id             char(30)     not null,  
    product_time            date        ,  
    product_level           char(10)    ,  
    product_name            varchar(200) ,  
    product_type1           varchar(20) ,  
    product_type2           char(10)    ,  
    product_monthly_sales_cnt integer    ,  
    product_comment_time    date        ,  
    product_comment_num     integer     ,  
    product_comment_content varchar(200)  
)  
SERVER gsmpp_server
```



```
OPTIONS(  
location 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',  
FORMAT 'CSV' ,  
DELIMITER ',',  
encoding 'utf8',  
header 'false',  
ACCESS_KEY 'access_key_value_to_be_replaced',  
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',  
fill_missing_fields 'true',  
ignore_extra_data 'true'  
)  
READ ONLY  
LOG INTO product_info_err  
PER NODE REJECT LIMIT 'unlimited';
```

If the following information is displayed, the foreign table has been created:

```
CREATE FOREIGN TABLE
```

7.2.5 Importing Data

Context

Before importing data, you are advised to optimize your design and deployment based on the following excellent practices, helping maximize system resource utilization and improving data import performance.

- In most cases, OBS data import performance is limited by concurrent network access rate. Therefore, you are advised to deploy multiple buckets on the OBS server to import data in parallel from buckets, better utilizing DN data transfer.
- Similar to the single table import, ensure that the I/O performance is greater than the maximum network throughput in the concurrent import.
- Set GUC parameters [raise_errors_if_no_files](#), [partition_mem_batch](#), and [partition_max_cache_size](#). Specify whether to distinguish the situation where the number of records in imported files is zero from the situation where an imported file does not exist when data is imported. Specify the number and size of data buffers.
- If a table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Procedure

- Step 1** Create a table in the GaussDB(DWS) database to store the data imported from the OBS. For details about the syntax, see CREATE TABLE.

The structure of the table must be consistent with that of the fields in the source data file. That is, the number of fields and field types must be the same. In addition, the structure of the target table must be the same as that of the foreign table. The field names can be different.

- Step 2** (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Step 3 Import data.

```
INSERT INTO [Target table name] SELECT * FROM [Foreign table name]
```

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see [Handling Import Errors](#).

```
INSERT 0 20
```

- If data fails to be loaded, troubleshoot the problem by following the instructions provided in [Handling Import Errors](#) and try again.

----End

Example

For example, create a table named *product_info*.

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price      integer      not null,
    product_id         char(30)     not null,
    product_time       date         ,
    product_level      char(10)     ,
    product_name       varchar(200) ,
    product_type1      varchar(20)  ,
    product_type2      char(10)     ,
    product_monthly_sales_cnt integer ,
    product_comment_time date       ,
    product_comment_num integer     ,
    product_comment_content varchar(200)
)
with (
    orientation = column,
    compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Run the following statement to import data from the *product_info_ext* foreign table to the *product_info* table:

```
INSERT INTO product_info SELECT * FROM product_info_ext;
```

7.2.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format error

When creating a foreign table, specify **LOG INTO *error_table_name***. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

```
SELECT * FROM error_table_name;
```

[Table 7-3](#) lists the columns of the *error_table_name* table.

Table 7-3 Columns in the error information table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where an error about data format occurs
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error
A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 7-4 Handling data import errors

Error Information	Cause	Solution
missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter () into the value of the second column, causing loss of the value of the third column. BE Belgium\ 1	1. If an error is reported due to missing columns, perform the following operations: • Add the r_reason_desc column to the source data file. • When creating a foreign table, set the parameter fill_missing_fields to on. In this way, if the last column of a row in the source data file is missing, it is set to NULL and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter noescaping to true when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter ignore_extra_data to on. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.

Error Information	Cause	Solution
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to numeric .
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the staff_id column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the SELECT statement with the DISTINCT keyword to ensure that all imported rows are unique. INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes.

7.2.7 Example

Step 1 On GaussDB(DWS), create the table (**tpcds.customer_address**) to store imported data.

```
CREATE TABLE tpcds.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)
)
WITH (orientation = column,compression=middle)
DISTRIBUTE BY hash (ca_address_sk);
```

- Step 2** Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input_data1** and **/input_data2** for storing data files, and create data directories **/input_data1/data** and **/input_data2/data**, respectively, in the two buckets.
- Step 3** Evenly divide the to-be-uploaded source data files into two parts, and upload them to the **/input_data1/data/** and **/input_data2/data/** folders on the OBS server, respectively.
- Step 4** On GaussDB(DWS), create a foreign table **tpcds.customer_address_ext** to receive data from the data server.

Assume that the OBS data server properly connects to the cluster network, the server IP address is *xxx.xxx.x.xx*, and the source data file format is CSV. In this case, plan the parallel import as described in this example.

Import information is set as follows:

- Because the source data file folders are **/input_data1/data/** and **input_data2/data/**, set **location** to **obs://input_data1/data/ | obs://input_data2/data/**.

Information about the data format is set based on the data format parameters specified in the source data file. The parameter settings are as follows:

- Set **format** to **CSV**.
- Set **encoding** to **UTF-8**.
- Set **delimiter** to **0E08**.
- Set **quote** to **0x1b**.
- encrypt** is set to **off**.
- Set **access_key** to the AK you have obtained.
- Set **secret_access_key** to the SK you have obtained.

 NOTE

The values of **access_key** and **secret_access_key** are examples only.

The import error tolerance settings are as follows:

- The maximum number (value of **PER NODE REJECT LIMIT**) of allowed data format errors on each DN is unlimited. That is, all the data format errors detected during data import are allowed.
- The data format errors detected during data import are recorded in the **customer_address_err** table by specifying **LOG INTO error_table_name**.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE tpcds.customer_address_ext
(
  ca_address_sk          integer          ,
  ca_address_id          char(16)         ,
  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)
)
SERVER gsmpp_server
OPTIONS(location 'obs://input_data1/data/ | obs://input_data2/data/',
FORMAT 'CSV' ,
encoding 'utf8',
DELIMITER E'\x08',
quote E'\x1b',
encrypt 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)LOG INTO customer_address_err PER NODE REJECT LIMIT 'unlimited';
```

Step 5 In GaussDB(DWS), import the data to the **tpcds.customer_address** table through the **tpcds.customer_address_ext** foreign table.

```
INSERT INTO tpcds.customer_address SELECT * FROM tpcds.customer_address_ext;
```

Step 6 Query data import errors in the **customer_address_err** table and rectify the errors. For details about error tables, see [Handling Import Errors](#).

```
SELECT * FROM customer_address_err;
```

Step 7 After data is imported, run the **ANALYZE** statement to generate table statistics.

```
ANALYZE tpcds.customer_address;
```

----End

7.3 Using GDS to Import Data from a Remote Server

7.3.1 Importing Data In Parallel Using GDS

INSERT and **COPY** statements can be used only for serially importing a small volume of data. To import a large volume of data to GaussDB(DWS), you can use GDS to import data in parallel through a foreign table.

Overview

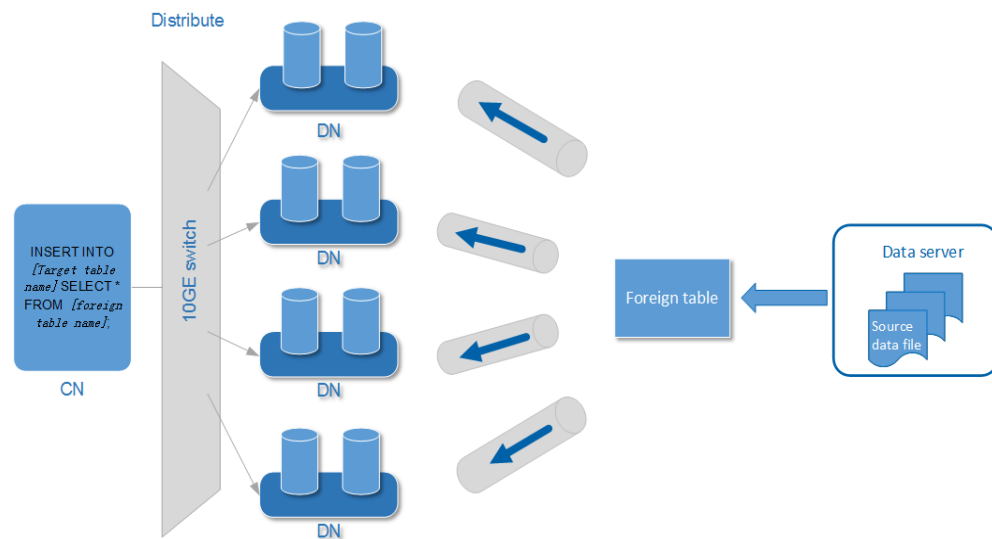
You can import data in parallel from the common file system (excluding HDFS) of a server to GaussDB(DWS).

Data files to be imported are specified based on the import policy and data formats set in a foreign table. Data is imported in parallel through multiple DNs from source data files to the database, which improves the overall data import performance. [Figure 7-4](#) shows an example.

- The CN only plans data import tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.
- In this way, the computing capacities and bandwidths of all the DNs are fully leveraged to import data, improving the import performance.

You can pre-process data (such as invalid character replacement and fault tolerance processing) by setting parameters in a foreign table.

Figure 7-4 Importing data in parallel



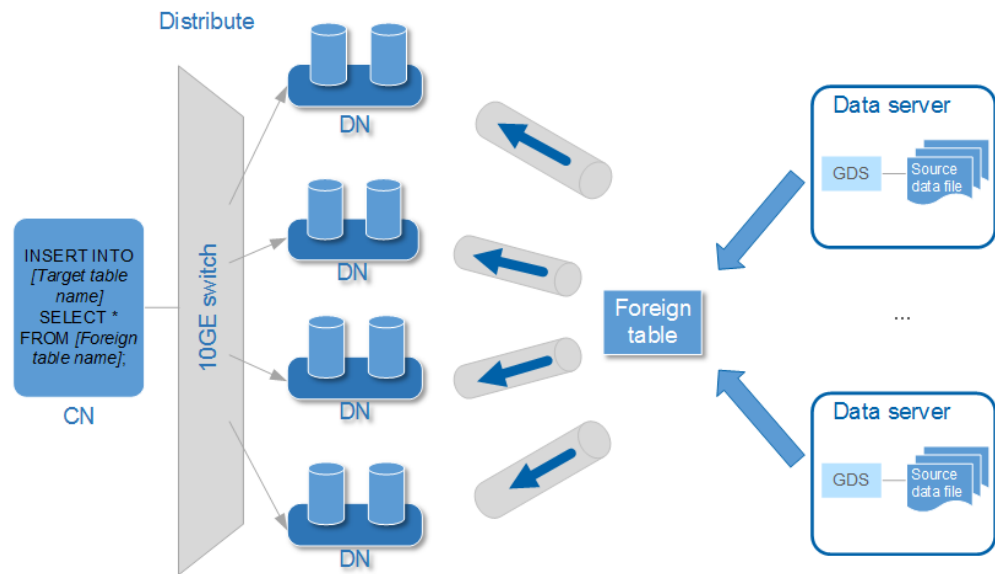
The concepts mentioned in the preceding figure are described as follows:

- **CN:** coordinator of GaussDB(DWS). After receiving import SQL requests from an application or client, the CN plans import tasks and delivers the tasks to DNs.
- **DN (Datanode):** data node of GaussDB(DWS). After receiving the import tasks delivered by the CN, DNs import data from the source data file to the target table in the database through a foreign table.
- **Source data file:** a file that stores data to be imported.
- **Data server:** a server that stores source data files. For security purposes, it is recommended that the data server and GaussDB(DWS) be on the same intranet.
- **Foreign table:** a table that stores information, such as the source location, format, destination location, encoding format, and data delimiter of a source data file. It is used to associate source data files with the target table.
- **Target table:** table in the database. It can be a row-store table or column-store table. Data in the data source files will be imported to this table.

Parallel Import Using GDS

- If a large volume of data is stored on multiple servers, deploy, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel, as shown in [Figure 7-5](#).

Figure 7-5 Parallel import from multiple data servers



NOTICE

The number of GDS processes cannot exceed that of DNs. If multiple GDS processes are connected to one DN, some of the processes will probably become abnormal.

- If data is stored on data servers, and both GaussDB(DWS) and the data servers have available I/O resources, you can use GDS for multi-thread concurrent import.

GDS determines the number of threads based on the number of concurrent import transactions. That is, even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

Multi-thread concurrent import enables you to:

- Make full use of resources and improve the concurrent import efficiency when you import multiple tables to the database.
- Speed up the import of a table with a large volume of data.

Table data is split into multiple data files, and multi-thread concurrent import is implemented by importing data using multiple foreign tables at the same time. Ensure that a data file can be read only by one foreign table.

Import Process

Figure 7-6 Concurrent import procedure

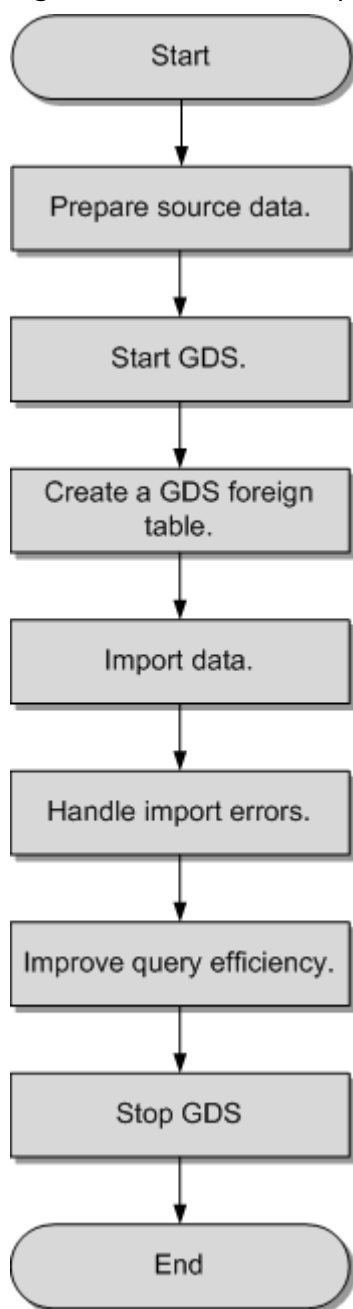


Table 7-5 Process description

Procedure	Description
Prepare source data.	Prepare the source data files to be imported to the database and upload the files to the data server. For details, see Preparing Source Data .

Procedure	Description
Start GDS.	Install, configure, and enable GDS on the data server. For details, see Installing, Configuring, and Starting GDS .
Create a foreign table.	A foreign table is used to identify source files. The foreign table stores information, such as the source location, format, destination location, encoding format, and inter-data delimiter of a source data file. For details, see Creating a GDS Foreign Table .
Import data.	After creating the foreign table, run the INSERT statement to quickly import data to the target table. For details, see Importing Data .
Handle the error table.	If errors occur during parallel data import, handle errors based on the error information to ensure data integrity. For details, see Handling Import Errors .
Improve query efficiency.	After data is imported, run the ANALYZE statement to generate table statistics. The ANALYZE statement stores the statistics in the pg_statistic system catalog. The execution plan generator uses the statistics to generate the most efficient query execution plan.
Stop GDS.	After data import is complete, log in to each data server and stop GDS. For details, see Stopping GDS .

7.3.2 Preparing Source Data

Scenarios

Generally, the data to be imported has been uploaded to the data server. In this case, you only need to check the communication between the data server and GaussDB(DWS), and record the data storage directory on the data server before the import.

If the data has not been uploaded to the data server, perform the following operations to upload it:

Procedure

Step 1 Log in to the data server as user **root**.

Step 2 Create the directory **/input_data**.

```
mkdir -p /input_data
```

Step 3 Upload the source data files to the created directory.

GDS parallel import supports source data only in CSV or TEXT format.

----End

7.3.3 Installing, Configuring, and Starting GDS

Scenarios

GaussDB(DWS) uses GDS to allocate the source data for parallel data import. Deploy GDS on the data server.

If a large volume of data is stored on multiple servers, deploy, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel. The procedure of installing, configuring, and starting GDS is the same on each data server. This section describes how to perform this procedure on one data server.

Context

1. GDS can be installed in the following OSs:

Kunpeng:

- Community Enterprise Operating System 7.6
- EulerOS 2.0 SP8
- Red Hat Enterprise Linux Server release 7.5
- NeoKylin 7.5/7.6

x86:

- SUSE Linux Enterprise Server 10 SP4 x86_64
- SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 x86_64
- SUSE Linux Enterprise Server 12 SP0/SP1/SP2/SP3 x86_64
- Red Hat Enterprise Linux Server release 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5 x86_64
- Community Enterprise Operating System 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4 x86_64

2. The GDS version must match the cluster version. For example, GDS V100R008C00 matches GaussDB(DWS) 1.3.X. Otherwise, the import or export may fail, or the import or export process may fail to respond.

Therefore, do not use the GDS of an earlier version. After the database is upgraded, download the GDS extracted from the GaussDB(DWS) software package as instructed in [Procedure](#). When the import or export starts, GaussDB(DWS) checks the GDS version. If the versions are inconsistent, an error message is displayed and the import or export is terminated.

To obtain the version number of GDS, run the following command in the GDS decompression directory:

```
gds -V
```

To view the database version, run the following SQL statement after connecting to the database:

```
SELECT version();
```

Procedure

- Step 1** Before using GDS to import or export data, perform the "Preparing an ECS as the GDS Data Server" and "Downloading the GDS Package and SSL Certificate" operations in [Step 1: Preparing an ECS as the GDS Server](#).

- Step 2** Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

```
mkdir -p /opt/bin/dws
```

- Step 3** Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_redhat_x64.zip** to the directory created in the previous step.

- Step 4** (Optional) If SSL is used, upload the SSL certificate to the directory created in [Step 2](#).

- Step 5** Go to the new directory and decompress the package.

```
cd /opt/bin/dws  
unzip dws_client_redhat_x64.zip
```

- Step 6** Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

```
groupadd gdsgrp  
useradd -g gdsgrp gds_user
```

- Step 7** Change the owner of the GDS package directory and source data file directory to the GDS user.

```
chown -R gds_user:gdsgrp /opt/bin/dws/gds  
chown -R gds_user:gdsgrp /input_data
```

- Step 8** Switch to user **gds_user**.

```
su - gds_user
```

- Step 9** Start the GDS.

GDS is green software and can be started after being decompressed. There are two ways to start GDS.

Method 1 is recommended when you only need to import data occasionally.

Method 2 is recommended when you need to import data regularly.

Method 1: Run the **gds** command to set startup parameters.

- Run the **gds** command to start GDS.
 - If data is transmitted in non-SSL mode, run the following command to start GDS:

```
gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
```

Example:

```
/opt/bin/dws/gds/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -  
l /opt/bin/dws/gds/gds_log.txt -D -t 2
```

- If data is transmitted in SSL mode, run the following command to start GDS:

```
gds -d dir -p ip:port -H address_string -l log_file -D  
-t worker_num --enable-ssl --ssl-dir Cert_file
```

Example:

Run the following command to upload the SSL certificate mentioned in [Step 4](#) to **/opt/bin**:

```
/opt/bin/dws/gds/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -  
l /opt/bin/dws/gds/gds_log.txt -D --enable-ssl --ssl-dir /opt/bin/
```

Replace the information in italic as required.

- **-d *dir***: directory storing data files that contain data to be imported. It is **/input_data/** in this tutorial.
- **-p *ip:port***: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with GaussDB(DWS). The listening port can be any one ranging from 1024 to 65535. The default port is **8098**. This parameter is set to **192.168.0.90:5000** in this tutorial.
- **-H *address_string***: network segment for hosts that can connect to and use GDS. The value must be in CIDR format. Set this parameter to enable GaussDB(DWS) to access GDS for data import. Ensure that the network segment covers all hosts in GaussDB(DWS).
- **-l *log_file***: GDS log directory and log file name. It is **/opt/bin/dws/gds/gds_log.txt** in this tutorial.
- **-D**: GDS in daemon mode. This parameter is used only in Linux.
- **-t *worker_num***: number of concurrent GDS threads. If the data server and GaussDB(DWS) have available I/O resources, you can increase the number of concurrent GDS threads.

GDS determines the number of threads based on the number of concurrent import transactions. That is, even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

- **--enable-ssl**: enables SSL for data transmission.
- **--ssl-dir *Cert_file***: SSL certificate directory. Set it to the certificate directory mentioned in [Step 4](#).
- For details about GDS parameters, see [gds Command Introduction](#).

Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start the GDS.

- Run the **gds_ctl.py** command to start GDS.
 - a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see [Table 7-6](#).

```
vim /opt/bin/gds/config/gds.conf
```

Example:

The **gds.conf** configuration file contains the following information:

```
<?xml version="1.0"?>
<config>
<gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/" err_dir="/err"
data_seg="100MB" err_seg="100MB" log_file="/log/gds_log.txt" host="10.10.0.1/24"
daemon='true' recursive="true" parallel="32"></gds>
</config>
```

Information in the configuration file is described as follows:

- The data server IP address is **192.168.0.90** and the GDS listening port is **5000**.
- Data files are stored in the **/input_data/** directory.
- Error log files are stored in the **/err** directory.
- The size of a single data file is 100 MB.

- The size of a single error log file is 100 MB.
- Logs are stored in the **/log/gds_log.txt** file.
- Only nodes with the IP address being 10.10.0.* can be connected.
- The GDS process is running in daemon mode.
- Recursive data file directories are used.
- The number of concurrent import threads is 32.

b. Start the GDS and check whether it has been started:

```
python gds_ctl.py start
```

Example:

```
cd /opt/bin/gds
python gds_ctl.py start
Start GDS gds1 [OK]
gds [options]:
-d dir      Set data directory.
-p port      Set GDS listening port.
  ip:port    Set GDS listening ip address and port.
-l log_file  Set log file.
-H secure_ip_range
              Set secure IP checklist in CIDR notation.          Required for GDS to start.
-e dir      Set error log directory.
-E size      Set size of per error log segment.(0 < size < 1024)
-S size      Set size of data segment.(1MB < size < 1024TB)
-t worker_num Set number of worker thread in multi-thread mode, the upper
limit is 32. If without t setting, the default value is 1.
-s status_file Enable GDS status report.
-D           Run the GDS as a daemon process.
-r           Read the working directory recursively.
-h           Display usage.
```

----End

gds.conf Parameter Description

Table 7-6 gds.conf configuration description

Attribute	Description	Value Range
name	Identifier	-
ip	Listening IP address	The IP address must be valid. Default value: 127.0.0.1
port	Listening port number	Value range: an integer ranging from 1024 to 65535 Default value: 8098
data_dir	Data file directory	-
err_dir	Error log file directory	Default value: data file directory
log_file	Log file path	-

Attribute	Description	Value Range
host	Sets the host allowed to be connected to GDS in CIDR format. Only a Linux OS is supported.	-
recursive	Whether the data file directories are recursive	Valid value: • true: recursive • false: not recursive Default value: false
daemon	Whether the process is running in the daemon mode.	Valid value: • true: The process is running in daemon mode. • false: The process is not running in daemon mode. Default value: false
parallel	Number of concurrent data import threads	Value range: 0 to 32 (integer) Default value: 32

7.3.4 Creating a GDS Foreign Table

The source data information and GDS access information are configured in a foreign table. Then, GaussDB(DWS) can import data from a data server to a database table based on the configuration in the foreign table.

Procedure

Step 1 Collect source data information and GDS access information.

You need to collect the following source data information:

- **format**: format of the data to be imported. Only data in CSV, TEXT, or FIXED format can be imported using GDS in parallel.
- **header**: whether a source data file has a header. This parameter is set only for files in CSV or FIXED format.
- **delimiter**: delimiter in the source data file. For example, it can be a comma (,).
- **encoding**: encoding format of the data source file. Assume that the encoding format is UTF-8.
- **eol**: line break character in the data file. It can be a default character, such as such as 0x0D0A or 0X0A, or a customized line break character, such as a string: !@#. This parameter can be set only for TEXT import.
- For details about other formats that can be identified by foreign tables, see data format parameters in CREATE FOREIGN TABLE (for GDS Import and Export).

You need to collect the following GDS access information:

location: GDS URL. GDS information in [Installing, Configuring, and Starting GDS](#) is used as an example. In non-SSL mode, **location** is set to **gsfs://192.168.0.90:5000/input_data/**. In SSL mode, **location** is set to **gsfss://192.168.0.90:5000/input_data/**. **192.168.0.90:5000** is the IP address and port number of GDS. **input_data** is the directory for storing source data files on the data server. Replace the values as required.

Step 2 Design an error tolerance mechanism for data import.

GaussDB(DWS) supports the following error tolerance in data import:

- **fill_missing_fields:** This parameter specifies whether to report an error when the last column in a row of the source data file is empty, or to fill the column with **NULL**.
- **ignore_extra_data:** This parameter specifies whether to report an error when the number of columns in the source data file is greater than that specified in the foreign table, or to ignore the extra columns.
- **per_node_reject_limit:** This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import will fail and an error message will be reported. You can also set it to **unlimited**.
- **compatible_illegal_chars:** This parameter specifies whether to report an error when an illegal character is encountered, or to convert it and proceed with the import.

The following describes the rules for converting an illegal character:

- **\0** is converted to a space.
- Other illegal characters are converted to question marks.
- If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to modify parameter settings that may cause import errors.
- **error_table_name:** This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- **remote_log 'name':** This parameter specifies whether to store data format errors in files on the GDS server. **name** is the prefix of the error data file.
- For details about error tolerance parameters, see error tolerance parameters in CREATE FOREIGN TABLE (for GDS Import and Export).

Step 3 After connecting to the database using **gsq**l or Data Studio, create a GDS foreign table based on the collected and design information.

For example:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server
OPTIONS
(
```

```
LOCATION 'gsfs://192.168.0.90:5000/input_data | gsfs://192.168.0.91:5000/input_data',  
FORMAT 'CSV',  
DELIMITER ',',  
ENCODING 'utf8',  
HEADER 'false',  
FILL_MISSING_FIELDS 'true',  
IGNORE_EXTRA_DATA 'true'  
)  
LOG INTO product_info_err  
PER NODE REJECT LIMIT 'unlimited';
```

The following describes information in the preceding command:

- The columns specified in the foreign table must be the same as those in the target table.
- Retain the value **gsmpp_server** for **SERVER**.
- Set **location** based on the GDS access information collected in [Step 1](#). If SSL is used, replace **gsfs** with **gsfss**.
- Set **FORMAT**, **DELIMITER**, **ENCODING**, and **HEADER** based on the source data information collected in [Step 1](#).
- Set **FILL_MISSING_FIELDS**, **IGNORE_EXTRA_DATA**, **LOG INTO**, and **PER NODE REJECT LIMIT** based on the error tolerance mechanism designed in [Step 2](#). **LOG INTO** specifies the name of the error table.

For details about the CREATE FOREIGN TABLE syntax, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

For more examples, see [Example](#).

- Example 1: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is CSV.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)  
)  
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',  
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 2: Create a GDS foreign table named **foreign_tpcds_reasons_SSL**. SSL is used and the data format is CSV.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons_SSL  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)  
)  
SERVER gsmpp_server OPTIONS (location 'gsfss://192.168.0.90:5000/* | gsfss://192.168.0.91:5000/*',  
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 3: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is TEXT.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)
```

```
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/*' | gsfs://192.168.0.91:5000/*',  
FORMAT 'TEXT', delimiter E'\x08', null '', reject_limit '2', EOL '0x0D') WITH err_foreign_tpcds_reasons;
```

7.3.5 Importing Data

This section describes how to create tables in GaussDB(DWS) and import data to the tables.

Before importing all the data from a table containing over 10 million records, you are advised to import some of the data, and check whether there is data skew and whether the distribution keys need to be changed (for details, see [Checking for Data Skew](#)). Troubleshoot the data skew if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Prerequisites

The GDS server can communicate with GaussDB(DWS).

- You need to create an ECS as the GDS server.
- The created ECS and GaussDB(DWS) must belong to the same region, VPC, and subnet.

Procedure

Step 1 Create a table in GaussDB(DWS) to store imported data. For details, see CREATE TABLE.

Step 2 (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the indexes of related tables before importing data. If the data uniqueness cannot be ensured, you are not advised to delete the unique indexes. You can create indexes again after the import is complete.

1. Assume that the ordinary index **product_idx** exists in the **product_id** column of the target table **product_info**. Delete the index from the table before importing data.

```
DROP INDEX product_idx;
```

2. After importing the data, create the **reasons_idx** index again.

```
CREATE INDEX product_idx ON product_info(product_id);
```

Step 3 Import data.

```
INSERT INTO [Target table name] SELECT * FROM [Foreign table name]
```

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see [Handling Import Errors](#).

```
INSERT 0 9
```

- If data fails to be loaded, troubleshoot the problem by following the instructions provided in [Handling Import Errors](#) and try again.

 NOTE

- If a data loading error occurs, the entire data import task will fail.
- Compile a batch-processing task script to concurrently import data. The degree of parallelism (DOP) depends on the server resource usage. You can test-import several tables, monitor resource utilization, and increase or reduce concurrency accordingly. Common resource monitoring commands include **top** for monitoring memory and CPU usage, **iostat** for monitoring I/O usage, and **sar** for monitoring networks. Examples are provided in [Example: Data Import Using Multiple Threads](#).
- If possible, more GDS servers can significantly improve the data import efficiency. Examples are provided in [Example: Parallel Import from Multiple Data Servers](#).
- In a scenario where many GDS servers import data concurrently, you can increase the TCP Keepalive interval for connections between GDS servers and DN's to ensure connection stability. (The recommended interval is 5 minutes.) TCP Keepalive settings of the cluster affect its fault detection response time.

----End

Example:

1. Create a target table named **reasons**.

```
CREATE TABLE reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
DISTRIBUTE BY HASH (r_reason_sk);
```
2. You are advised to delete the indexes from the target table before the import. Assume that an ordinary table index **reasons_idx** exists on the **r_reason_id** column in the **reasons** table. Delete the index before the import. Delete the index from the table.

```
DROP INDEX reasons_idx;
```
3. Import data from source data files through the **foreign_tpcds_reasons** foreign table to the **reasons** table.

```
INSERT INTO reasons SELECT * FROM foreign_tpcds_reasons ;
```
4. You can create indexes again after the import is complete.

```
CREATE INDEX reasons_idx ON reasons(r_reason_id);
```

7.3.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format error
When creating a foreign table, specify **LOG INTO error_table_name**. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

```
SELECT * FROM error_table_name;
```

Table 7-7 lists the columns of the *error_table_name* table.

Table 7-7 Columns in the error information table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where an error about data format occurs
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 7-8 Handling data import errors

Error Information	Cause	Solution
missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter () into the value of the second column, causing loss of the value of the third column. BE Belgium\ 1	1. If an error is reported due to missing columns, perform the following operations: • Add the r_reason_desc column to the source data file. • When creating a foreign table, set the parameter fill_missing_fields to on. In this way, if the last column of a row in the source data file is missing, it is set to NULL and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter noescaping to true when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter ignore_extra_data to on. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.

Error Information	Cause	Solution
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to numeric .
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the staff_id column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the SELECT statement with the DISTINCT keyword to ensure that all imported rows are unique. INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes.

7.3.7 Stopping GDS

Scenarios

Stop GDS after data is imported successfully.

Procedure

Step 1 Log in as user **gds_user** to the data server where GDS is installed.

Step 2 Select the mode of stopping GDS based on the mode of starting it.

- If GDS is started using the **gds** command, perform the following operations to stop GDS:

a. Query the GDS process ID:

```
ps -ef|grep gds
```

For example, the GDS process ID is 128954.

```
ps -ef|grep gds
```

```
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/
```

```
gds_log.txt -D
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
```

- b. Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.

```
kill -9 128954
```

- If GDS is started using the **gds_ctl.py** command, perform the following operations to stop GDS:

```
cd /opt/bin/dws/gds
python gds_ctl.py stop
```

----End

7.3.8 Example

Example: Parallel Import from Multiple Data Servers

The data servers reside on the same intranet as the cluster. Their IP addresses are 192.168.0.90 and 192.168.0.91. Source data files are in CSV format.

1. Create the target table **tpcds.reasons**.

```
CREATE TABLE tpcds.reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```

2. Log in to each GDS data server as user **root** and create the **/input_data** directory for storing data files on the servers. The following takes the data server whose IP address is 192.168.0.90 as an example. Operations on the other server are the same.

```
mkdir -p /input_data
```

3. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

4. Evenly distribute source data files to the **/input_data** directories on the data servers.

5. Change the owners of source data files and the **/input_data** directory on each data server to **gds_user**. The data server with the IP address 192.168.0.90 is used as an example.

```
chown -R gds_user:gdsgrp /input_data
```

6. Log in to each data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/gds**. Source data files are stored in **/input_data/**. The IP addresses of the data servers are 192.168.0.90 and 192.168.0.91. The GDS listening port is 5000. GDS runs in daemon mode.

Start GDS on the data server whose IP address is 192.168.0.90.

```
/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

Start GDS on the data server whose IP address is 192.168.0.91.

```
/gds/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24 -D
```

7. Create the foreign table **tpcds.foreign_tpcds_reasons** for the source data.

Set import mode parameters as follows:

- Set the import mode to **Normal**.

- When GDS is started, the source data file directory is **/input_data** and the GDS listening port is 5000. Therefore, set **location** to **gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/***.

Information about the data format is set based on data format parameters specified during data export. The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **0x1b**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to a double quotation mark.
- **header** is set to **false**, indicating that the first row is regarded as a data row when a file is imported.

Set import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons** table.

Based on the above settings, the foreign table is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields
'false') LOG INTO err_tpcds_reasons PER NODE REJECT LIMIT 'unlimited';
```

8. Import data through the foreign table **tpcds.foreign_tpcds_reasons** to the target table **tpcds.reasons**.

```
INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
```
9. Query data import errors in the **err_tpcds_reasons** table and rectify the errors (if any). For details, see [Handling Import Errors](#).

```
SELECT * FROM err_tpcds_reasons;
```
10. After data import is complete, log in to each data server as user **gds_user** and stop GDS.

The data server with the IP address 192.168.0.90 is used as an example. The GDS process ID is 128954.

```
ps -ef|grep gds
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
kill -9 128954
```

Example: Data Import Using Multiple Threads

The data server resides on the same intranet as the cluster. The server IP address is 192.168.0.90. Source data files are in CSV format. Data will be imported to two tables using multiple threads in **Normal** mode.

1. In the database, create the target tables **tpcds.reasons1** and **tpcds.reasons2**.

```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
CREATE TABLE tpcds.reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```
2. Log in to the GDS data server as user **root**, and then create the data file directory **/input_data** and its sub-directories **/input_data/import1/** and **/input_data/import2/**.

```
mkdir -p /input_data
```
3. Store the source data files of the target table **tpcds.reasons1** in **/input_data/import1/** and the source data files of the target table **tpcds.reasons2** in **/input_data/import2/**.
4. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```
5. Change the owners of source data files and the **/input_data** directory on the data server to **gds_user**.

```
chown -R gds_user:gdsgrp /input_data
```
6. Log in to the data server as user **gds_user** and start GDS.
The GDS installation path is **/gds**. Source data files are stored in **/input_data/**. The IP address of the data server is 192.168.0.90. The GDS listening port is 5000. GDS runs in daemon mode. The degree of parallelism is 2. A recursive directory is specified.

```
/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2 -r
```
7. In the database, create the foreign tables **tpcds.foreign_tpcds_reasons1** and **tpcds.foreign_tpcds_reasons2** for the source data.
The foreign table **tpcds.foreign_tpcds_reasons1** is used as an example to describe how to set parameters in a foreign table.
Set import mode parameters as follows:
 - Set the import mode to **Normal**.
 - When GDS is started, the configured source data file directory is **/input_data** and the GDS listening port is 5000. However, source data files are actually stored in **/input_data/import1/**. Therefore, set **location** to **gsfs://192.168.0.90:5000/import1/***.Information about the data format is set based on data format parameters specified during data export. The parameter settings are as follows:
 - **format** is set to **CSV**.
 - **encoding** is set to **UTF-8**.
 - **delimiter** is set to **E'\x08'**.
 - **quote** is set to **0x1b**.
 - **null** is set to an empty string without quotation marks.
 - **escape** is set to a double quotation mark.

- **header** is set to **false**, indicating that the first row is regarded as a data row when a file is imported.

Set import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons1**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons1** table.
- If the last column (**fill_missing_fields**) in a source data file is missing, the **NULL** column will be automatically added to the target file.

Based on the above settings, the foreign table **tpcds.foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import1/*', format 'CSV', mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields 'on') LOG INTO
err_tpcds_reasons1 PER NODE REJECT LIMIT 'unlimited';
```

Based on the above settings, the foreign table **tpcds.foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import2/*', format 'CSV', mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields 'on') LOG INTO
err_tpcds_reasons2 PER NODE REJECT LIMIT 'unlimited';
```

8. Import data through the foreign table **tpcds.foreign_tpcds_reasons1** to **tpcds.reasons1** and through **tpcds.foreign_tpcds_reasons2** to **tpcds.reasons2**.

```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```

9. Query data import errors in the **err_tpcds_reasons1** and **err_tpcds_reasons2** tables and rectify the errors (if any). For details, see [Handling Import Errors](#).

```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```

10. After data import is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is 128954.

```
ps -ef|grep gds
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D -t 2 -r
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
kill -9 128954
```

7.4 Running the INSERT Statement to Insert Data

Run the **INSERT** statement to write data into GaussDB(DWS) in either of the following ways:

- Use the client tool provided by GaussDB(DWS) to write data into it.
For details, see section "Inserting Data to a Table."

- Connect to the database using the JDBC or ODBC driver and run the **INSERT** statement to write data to GaussDB(DWS).

You can add, modify, and delete database transactions for GaussDB(DWS). **INSERT** is the simplest way to write data. It is applicable to scenarios with small data volume and low concurrency.

7.5 Running the COPY FROM STDIN Statement to Import Data

7.5.1 Data Import Using COPY FROM STDIN

This method is applicable to low-concurrency scenarios where a small volume of data is to be imported.

Run the **COPY FROM STDIN** statement to import data to GaussDB(DWS) in either of the following ways:

- Write data into GaussDB(DWS) by typing.
- Import data from a file or database to GaussDB(DWS) through the CopyManager interface driven by JDBC. You can use any parameters in the **COPY** syntax.

7.5.2 Introduction to the CopyManager Class

CopyManager is an API interface class provided by the JDBC driver in GaussDB(DWS). It is used to import data to GaussDB(DWS) in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **org.postgresql.copy** package class and is inherited from the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Constructor Method

```
public CopyManager(BaseConnection connection)

throws SQLException
```

Basic Methods

Table 7-9 Basic methods of CopyManager

Return Value	Method	Description	Throws
CopyIn	copyIn(String sql)	-	SQLException

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

7.5.3 Example 1: Importing and Exporting Data Through Local Files

When the JAVA language is used for secondary development based on GaussDB(DWS), you can use the CopyManager interface 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(DWS) JDBC driver before executing it.

```
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://localhost:8000/postgres"); //URL of the database
        String username = new String("username"); //User name
        String password = new String("passwd"); //Password
        String tablename = new String("migration_table"); //Define table information.
        String tablename1 = new String("migration_table_1"); //Define 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 data from the migration_table table to the d:/data.txt file.
        try {
            copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            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) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

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

```
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 with (" + "DELIMITER" + "" + delimiter +
"" + "ENCODING " + "" + encoding + "" + ")", fileInputStream);
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

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();
            }
        }
    }
}
}
```

7.5.4 Example 2: Migrating Data from a MySQL Database to GaussDB(DWS)

The following example shows how to use CopyManager to migrate data from a MySQL database to GaussDB(DWS).

```
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://localhost:8000/postgres"); //URL of the database
        String user = new String("username"); //GaussDB(DWS) user name
        String pass = new String("passwd"); //GaussDB(DWS) password
        String tablename = new String("migration_table_1"); //Define table information.
        String delimiter = new String("|"); //Define a delimiter.
        String encoding = new String("UTF8"); //Define a character set.
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer(); //Define a 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 with (DELIMITER " + "'" + delimiter + "'" + "," + " "
ENCODING " + "'" + encoding + "'");

        //Submit 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();
        Connection conn = DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "Gauss@123");
        Statement stmt = conn.createStatement();
        rs = stmt.executeQuery("select * from migration_table");
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return rs;
}
}
```


7.6 Using a gsql Meta-Command to Import Data

The **gsql** tool of GaussDB(DWS) provides the **\copy** meta-command to import data.

\copy Command

For details about the **\copy** command, see [Table 7-10](#).

Table 7-10 \copy meta-command

Syntax	Description
<code>\copy { table [(column_list)] (query) } { from to } { filename stdin stdout pstdin pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list *] [force not null column_list]]</code>	You can run this command to import or export data after logging in to the database on any gsql client. Different from the COPY statement in SQL, this command performs read/write operations on local files rather than files on database servers. The accessibility and permissions of the local files are restricted to local users. NOTE \copy only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or COPY is preferred for data import.

Parameter Description

- **table**
Specifies the name (possibly schema-qualified) of an existing table.
Value range: an existing table name.
- **column_list**
Specifies an optional list of columns to be copied.
Value range: any field in the table. If the column list is not specified, all columns in the table will be copied.
- **query**
Specifies that the results will be copied.
Valid value: a **SELECT** or **VALUES** command in parentheses.
- **filename**
Specifies the absolute path of a file. To run the **\copy** command, the user must have the write permission for this path.
- **stdin**
Specifies that input comes from the client application.
- **stdout**

Specifies that output goes to the client application.

- `pstdin`

Specifies that input comes from the `gsq` client.

- `pstdout`
- Specifies that output goes to the `gsq` client.
- `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**. After specifying **BINARY**, **CSV**, **FIXED** and **TEXT** cannot be specified through **option** or **copy_option**.

- `oid`

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.

Valid value: **true**, **on**, **false**, and **off**.

Default value: **false**

- `delimiter [ as ] 'character'`

Specifies the character that separates columns within each row (line) of the file.

 NOTE

- A delimiter cannot be `\r` or `\n`.
- A delimiter cannot be the same as the **null** value. The delimiter of CSV data cannot be same as the **quote** value.
- The delimiter of TEXT data cannot contain any of the following characters:
`\.abcdefghijklmnopqrstuvwxyz0123456789`
- 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-characters and invisible characters for delimiters. For example, you can use multi-characters such as `'$^&';`, and invisible characters such as `0x07`, `0x08`, and `0x1b`.

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 [ as ] 'string'`

Specifies that a string represents a null value in a data file.

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:

- An empty string without quotation marks in CSV format

- \N in TEXT format
- header

Specifies whether a data file contains a table header. **header** is available only for CSV and FIXED files.

In data import scenarios, 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.

If header is **on**, **fileheader** must be specified. **fileheader** specifies the content in the header. If header is **off**, the exported file does not contain a header.

Valid value: **true**, **on**, **false**, and **off**.

Default value: **false**
- quote [as] 'character'

Specifies the quote character for a CSV file.

Default value: double quotation mark (").

 **NOTE**

 - The **quote** value cannot be the same as the **delimiter** or **null** value.
 - The **quote** value must be a single-byte character.
 - Invisible characters are recommended, such as 0x07, 0x08, and 0x1b.
- escape [as] 'character'

Specifies the escape character in CSV format. The value must be a single-byte character.

Default value: double quotation mark ("). If the value is the same as the **quote** value, it will be replaced with \0.
- force quote column_list | *

In CSV COPY TO mode, forces quoting to be used for all not-null values in each specified column. NULL will not be quoted.

Value range: an existing column.
- force not null column_list

In CSV COPY FROM mode, processes each specified column as though it were quoted and hence not a null value.

Value range: an existing column.

Examples

1. Create a target table, **a**.

```
CREATE TABLE a(a int);
```
2. Import data.
 - a. Copy data from **stdin** to table **a**.

```
\copy a from stdin;
```

When the >> characters are displayed, enter data. Enter a backslash and a period (\.) to end your input.

Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

```
>> 1
>> 2
>> \.
```

Query data imported to table **a**.

```
SELECT * FROM a;  
a  
---  
1  
2  
(2 rows)
```

- b. Copy data from a local file to table **a**. Assume that the local file is **/home/omm/2.csv**.
- Commas (,) are used as delimiters.
 - If the number of columns defined in the source data file is greater than that in a foreign table, extra columns will be ignored during import.

```
\copy a FROM '/home/omm/2.csv' WITH (delimiter',';IGNORE_EXTRA_DATA 'on');
```

7.7 Importing Data From MRS to a Data Warehouse Cluster

7.7.1 Overview of Importing Data from MRS

MapReduce Service (MRS) is a big data cluster running based on the open source Hadoop ecosystem. It provides the capabilities for storing and analyzing massive sets of data, which satisfies user data storage and processing requirements. For details, see [MapReduce Service User Guide](#).

You can use Hive/Spark (analysis cluster of MRS) to store massive volumes of service data. Hive/Spark data files are stored on HDFS. On GaussDB(DWS), you can connect a GaussDB(DWS) cluster to an MRS cluster, read data from HDFS files, and write the data to GaussDB(DWS) when the clusters are on the same network.

Importing Data From MRS to a GaussDB(DWS) Cluster

1. [Preparing Data in an MRS Cluster](#)
2. (Optional) [Manually Creating a Foreign Server](#)
3. [Creating a Foreign Table](#)
4. [Importing Data](#)
5. [Deleting Resources](#)

7.7.2 Preparing Data in an MRS Cluster

Before importing data from MRS to a GaussDB(DWS) cluster, you must have:

1. Created an MRS cluster.
2. Created the Hive/Spark ORC table in the MRS cluster and stored the table data to the HDFS path corresponding to the table.

If you have completed the preparations, skip this section.

In this tutorial, the Hive ORC table will be created in the MRS cluster as an example to complete the preparation work. The process for creating the Spark ORC table in the MRS cluster and the SQL syntax are similar to those of Hive.

Data File

The sample data of the **product_info.txt** data file is as follows:

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very
good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really
super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The
seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The
clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow
the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really
amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open
the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,
2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,
112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The
clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,
37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-L-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The
clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,
3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The
seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,
6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I
like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,
2056,2017-10-02,3842,very good
```

Creating a Hive ORC Table in an MRS Cluster

1. Create an MRS cluster.

For details, see [Creating an MRS Data Source Connection](#) in the Data Warehouse Service Management Guide.

2. Log in to the Hive client of the MRS cluster.

- a. Log in to a Master node.

For details, see [Logging In to an ECS](#) in the *MapReduce Service User Guide*.

- b. Run the following command to switch the user:

```
sudo su - omm
```

- c. Run the following command to go to the client directory:

```
cd /opt/client
```

- d. Run the following command to configure the environment variables:

```
source bigdata_env
```

- e. If Kerberos authentication has been enabled for the current cluster, run the following command to authenticate the current user, which requires that the current user has the permission to create Hive tables. Next, configure a role with the corresponding permissions. For details, see [Creating a Role](#) in the *MapReduce Service User Guide*. Then, bind a role to a user. For details, see [Creating a User](#) in the *MapReduce Service User Guide*. If the Kerberos authentication is disabled for the current cluster, skip this step.

```
kinit MRS cluster user
```

Example: **kinit hiveuser**

- f. Run the following command to start the Hive client:

```
beeline
```

3. Create a database demo on Hive.

Run the following command to create the database demo:

```
CREATE DATABASE demo;
```

4. Create table **product_info** of the **Hive TEXTFILE** type in the database demo and import the [Data File](#) (**product_info.txt**) to the HDFS path corresponding to the table.

Run the following command to switch to the database demo:

```
USE demo;
```

Run the following command to create table **product_info** and define the table fields based on data in the [Data File](#).

```
DROP TABLE product_info;
```

```
CREATE TABLE product_info
(
  product_price      int      not null,
  product_id         char(30) not null,
  product_time       date      ,
  product_level      char(10)  ,
  product_name       varchar(200) ,
  product_type1      varchar(20) ,
  product_type2      char(10)   ,
  product_monthly_sales_cnt int   ,
  product_comment_time date     ,
  product_comment_num int      ,
  product_comment_content varchar(200)
)
row format delimited fields terminated by ','
stored as TEXTFILE;
```

For details about how to import data to an MRS cluster, see [Managing Active Clusters > Managing Data Files](#) in the *MapReduce Service User Guide*.

5. Create a Hive ORC table named **product_info_orc** in the database demo.

Run the following command to create the Hive ORC table **product_info_orc**. The table fields are the same as those of the **product_info** table created in the previous step.

```
DROP TABLE product_info_orc;
```

```
CREATE TABLE product_info_orc
(
  product_price      int      not null,
  product_id         char(30) not null,
  product_time       date      ,
  product_level      char(10)  ,
```

```
product_name      varchar(200) ,
product_type1     varchar(20) ,
product_type2     char(10) ,
product_monthly_sales_cnt int ,
product_comment_time date ,
product_comment_num int ,
product_comment_content varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

6. Insert data in the **product_info** table to the Hive ORC table **product_info_orc**.
`insert into product_info_orc select * from product_info;`

Query table **product_info_orc**.

```
select * from product_info_orc;
```

If data displayed in the [Data File](#) can be queried, the data has been successfully inserted to the ORC table.

7.7.3 Manually Creating a Foreign Server

In the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)** for creating a foreign table, you need to specify a foreign server associated with the MRS data source connection.

When you create an MRS data source connection on the GaussDB(DWS) management console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you want to create a foreign table in the default database **postgres** to read MRS data, skip this section.

If you want to use a common user to create a foreign table in a customized database to read MRS data, you must manually create a foreign server in the customized database. This section describes how to create a foreign server in a customized database as a common user. The procedure is as follows:

1. Ensure that an MRS data source connection has been created for the GaussDB(DWS) cluster.
For details, see [Creating an MRS Data Source Connection](#) in the Data Warehouse Service Management Guide.
2. [Creating a User and a Database and Granting the User Foreign Table Permissions](#)
3. [Manually Creating a Foreign Server](#)

NOTE

If you no longer need to read data from the MRS data source and have deleted the MRS data source on the GaussDB(DWS) management console, only the foreign server automatically created in the default database **postgres** will be deleted, and the manually created foreign server needs to be deleted manually. For details about the deletion, see [Deleting the Manually Created Foreign Server](#).

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD "Bigdata@123";
```

Switch to the created user.

```
SET ROLE dbuser PASSWORD "Bigdata@123";
```

Run the following command to create a database:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

datname	datdba	encoding	datcollate	datctype	datistemplate	dataallowconn	datconnlimit	datacl
template1	10	0	C	C	t	t	-1	14146 1351
template0	10	0	C	C	t	f	-1	14146 1350
postgres	10	0	C	C	f	t	-1	14146 1352
mydatabase	17000	0	C	C	f	t	-1	14146 1351

(4 rows)

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password as prompted.

NOTE

Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:


```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
       r.rolcreatorole, r.rolcreatedb, r.rolcanlogin,
       r.rolconlimit, r.rolvalidbegin, r.rolvaliduntil,
       ARRAY(SELECT b.rolname
              FROM pg_catalog.pg_auth_members m
              JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
              WHERE m.member = r.oid) as memberof
       , r.rolreplication
       , r.rolauditadmin
       , r.rolsystemadmin
       , r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

rolname	rolsuper	rolinherit	rolcreatorole	rolcreatedb	rolcanlogin	rolconlimit	rolvalidbegin	rolvaliduntil	memberof	rolreplication	rolauditadmin	rolsystemadmin	roluseft
dbuser	f	t	f	t	t	-1			{}	f			
lily	f	t	f	f	t	-1			{}	f			
omm	t	t	t	t	t	-1			{}				
t	t	t	t										

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin;
```

Enter the password as prompted.

- If you have not logged in to the **gsql** client or have exited the **gsql** client by running the **\q** command, run the following command to reconnect to it:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter the password as prompted.

Step 2 Run the following command to query the information about the foreign server that is automatically created:

```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
srvoptions					
gsmpp_server	10	13673			
gsmpp_errorinfo_server	10	13678			
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca	16476	13685			
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(3 rows)					

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in [Creating a User and a Database and Granting the User Foreign Table Permissions](#) to connect to **mydatabase** created by the user:

```
\c mydatabase dbuser;
```

Step 4 Create a foreign server.

For details about the syntax for creating foreign servers, see CREATE SERVER. For example:

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA WRAPPER
HDFS_FDW
OPTIONS
(
address '192.168.1.245:25000,192.168.1.218:25000',
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*
You can customize a name.
In this example, specify the name to the value of the **srvname** field recorded in [Step 2](#), such as **hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca**.
Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.
- **FOREIGN DATA WRAPPER**
This parameter can only be set to **HDFS_FDW**, which already exists in the database.

- **OPTIONS** parameters

Set the following parameters to the values under **srvoptions** recorded in [Step 2](#).

- address

Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.

- hdfsconfigpath

Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.

- type

Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE  
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

The server is successfully created if the returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
srvoptions					
-----+-----+-----+-----+-----+-----					
+-----+-----+-----+-----+-----+-----					
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca		16476	13685		
{\"address=192.168.1.245:25000,192.168.1.218:25000\",hdfsconfigpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(1 row)					

----End

7.7.4 Creating a Foreign Table

This section describes how to create a Hadoop foreign table in the GaussDB(DWS) database to access the Hadoop structured data stored on MRS HDFS. A Hadoop foreign table is read-only. It can only be queried using **SELECT**.

You can perform the following steps to create a foreign table:

1. Ensure that you have completed preparations in [Prerequisites](#).
2. According to the description of the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**, you need to obtain the following information:
 - a. [Obtaining the HDFS Path of the MRS Data Source](#)
 - b. [Obtaining Information About the Foreign Server Connected to the MRS Data Source](#)
3. [Creating a Foreign Table](#)

Reference information: [Data Type Conversion](#)

Prerequisites

- You have created an MRS cluster and imported data to the ORC table in the Hive/Spark database.
For details, see [Preparing Data in an MRS Cluster](#).
- You have created an MRS data source connection for the GaussDB(DWS) cluster.

For details, see [Creating an MRS Data Source Connection](#) in the Data Warehouse Service Management Guide.

Obtaining the HDFS Path of the MRS Data Source

There are two methods for you to obtain the HDFS path.

- **Method 1**

For Hive data, log in to the Hive client of MRS (see [2](#)), run the following command to view the detailed information about the table, and record the data storage path in the **location** parameter:

```
use <database_name>;  
desc formatted <table_name>;
```

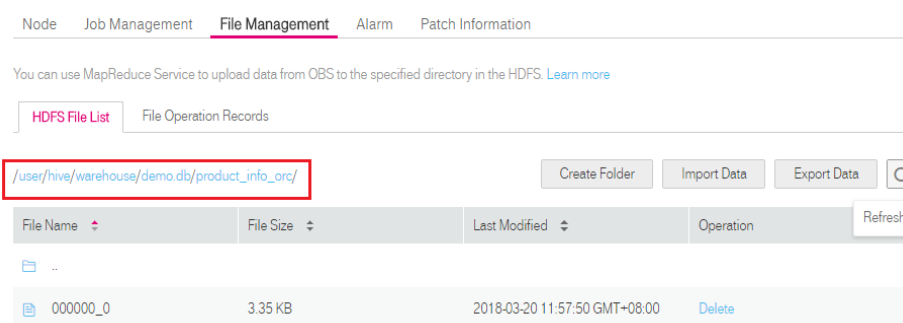
For example, if the value of the **location** parameter in the returned result is **hdfs://hacluster/user/hive/warehouse/demo.db/product_info_orc/**, the HDFS path is **/user/hive/warehouse/demo.db/product_info_orc/**.

- **Method 2**

Perform the following steps to obtain the HDFS path:

- a. Log in to the MRS management console.
- b. Choose **Cluster > Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.
- c. Click **File Management** and select **HDFS File List**.
- d. Go to the storage directory of the data to be imported to the GaussDB(DWS) cluster and record the path.

Figure 7-7 Checking the data storage path on MRS



Obtaining Information About the Foreign Server Connected to the MRS Data Source

Step 1 Use the user who creates the foreign server to connect to the corresponding database.

Determine whether to use a common user to create a foreign table in the customized database based on requirements.

- **Yes**

- a. Ensure that you have created common user **dbuser** and **mydatabase**, and manually created a foreign server in **mydatabase** by following steps in [Manually Creating a Foreign Server](#).

- b. Connect to the database **mydatabase** as user **dbuser** through the database client tool provided by GaussDB(DWS).

If you have connected to the database using the **gsql** client, run the following command to switch the user and database:

```
\c mydatabase dbuser;
```

Enter your password as prompted.

- **No**

When you create an MRS data source connection on the GaussDB(DWS) management console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. To create a foreign table in the default database **postgres** as a database administrator (**dbadmin**), you need to use the database client tool provided by GaussDB(DWS) to connect to the database. For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

Step 2 Run the following command to view the information about the created foreign server connected to the MRS data source:

```
SELECT * FROM pg_foreign_server;
```

 NOTE

You can also run the **\desc+** command to view the information about the foreign server.

The returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
srvoptions					
gsmpp_server	10	13673			
gsmpp_errorinfo_server	10	13678			
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca	16476	13685			
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(3 rows)					

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

----End

Creating a Foreign Table

After [Obtaining Information About the Foreign Server Connected to the MRS Data Source](#) and [Obtaining the HDFS Path of the MRS Data Source](#) are completed, you can create a foreign table to read data from the MRS data source.

The syntax for creating a foreign table is as follows. For details, see the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...] } ] |
  table_constraint [, ...] [, ...] } )
SERVER dfs_server
OPTIONS ( { option_name ' value ' } [, ...] )
DISTRIBUTE BY { ROUNDROBIN | REPLICATION }
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *foreign_product_info*, set parameters in the syntax as follows:

- **table_name**

Mandatory. This parameter specifies the name of the foreign table to be created.

- Table column definitions

- **column_name**: specifies the name of a column in the foreign table.
- **type_name**: specifies the data type of the column.

Multiple columns are separate by commas (,).

The number of columns and column types in the foreign table must be the same as those in the data stored on MRS. Learn [Data Type Conversion](#) before defining column data types.

- **SERVER dfs_server**

This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign table can read data from an MRS cluster by configuring the foreign server and connecting to the MRS data source.

Enter the value of the **srvname** field queried in [Obtaining Information About the Foreign Server Connected to the MRS Data Source](#).

- **OPTIONS** parameters

These are parameters associated with the foreign table. The key parameters are as follows:

- **format**: This parameter is mandatory. The value can only be **orc**. It specifies the format of the source data file. Only Hive ORC files are supported.
- **foldername**: This parameter is mandatory. It specifies the HDFS directory for storing data or data file path.

If the MRS analysis cluster has enabled Kerberos authentication, ensure that the MRS user having the MRS data source connection has the read and write permissions for the directory.

Follow the steps in [Obtaining the HDFS Path of the MRS Data Source](#) to obtain the HDFS path, which is the value of parameter **foldername**.

- **encoding**: This parameter is optional. It specifies the encoding format of a source data file in the foreign table. Its default value is **utf8**.
- **DISTRIBUTE BY**

This parameter specifies the data read mode for the foreign table. There are two read modes supported. In this example, **ROUNDROBIN** is selected.

- **ROUNDROBIN:** When a foreign table reads data from the data source, each node in a GaussDB(DWS) cluster randomly reads some data and integrates the random data to a complete data set.
- **REPLICATION:** When a foreign table reads data from the data source, each node in the GaussDB(DWS) cluster reads a complete data set.
- Other parameters in the syntax

Other parameters are optional. You can set them as required. In this example, you do not need to set these parameters. For details, see **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

Based on the above settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

CREATE FOREIGN TABLE foreign_product_info
(
    product_price          integer      not null,
    product_id             char(30)    not null,
    product_time           date        ,
    product_level          char(10)    ,
    product_name           varchar(200) ,
    product_type1          varchar(20) ,
    product_type2          char(10)    ,
    product_monthly_sales_cnt integer    ,
    product_comment_time   date        ,
    product_comment_num    integer    ,
    product_comment_content varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
    format 'orc',
    encoding 'utf8',
    foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

Data Type Conversion

Data is imported to Hive/Spark and then stored on HDFS in ORC format. Actually, GaussDB(DWS) reads ORC files on HDFS, and queries and analyzes data in these files.

Data types supported by Hive/Spark are different from those supported by GaussDB(DWS). Therefore, you need to learn the mapping between them. [Table 7-11](#) describes the mapping in detail.

Table 7-11 Data type mapping

Type	Column Type Supported by an HDFS/OBS Foreign Table of GaussDB(DWS)	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in two bytes	SMALLINT	SMALLINT	SMALLINT

Type	Column Type Supported by an HDFS/OBS Foreign Table of GaussDB(DWS)	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in four bytes	INTEGER	INT	INT
Integer in eight bytes	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE	FLOAT
Scientific data type	DECIMAL[p (,s)] The maximum precision can reach up to 38.	DECIMAL The maximum precision can reach up to 38 (Hive 0.11).	DECIMAL
Date type	DATE	DATE	DATE
Time type	TIMESTAMP	TIMESTAMP	TIMESTAMP
BOOLEAN type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)	STRING
VARCHAR type	VARCHAR(n)	VARCHAR (n)	VARCHAR (n)
String	TEXT(CLOB)	STRING	STRING

7.7.5 Importing Data

Viewing Data in the MRS Data Source by Directly Querying the Foreign Table

If the data amount is small, you can directly run SELECT to query the foreign table and view the data in the MRS data source.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM foreign_product_info;
```

If the query result displays data specified in [Data File](#), the import is successful. The following information is displayed at the end of the query result:

(20 rows)

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

You can query the MRS data after importing it to GaussDB(DWS).

Step 1 Create a table in GaussDB(DWS) to store imported data.

The target table structure must be the same as the structure of the foreign table created in [Creating a Foreign Table](#). That is, the number of fields and field types must be the same.

For example, create a table named **product_info**. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
  product_price      integer      not null,
  product_id         char(30)     not null,
  product_time       date         ,
  product_level      char(10)     ,
  product_name       varchar(200) ,
  product_type1      varchar(20)  ,
  product_type2      char(10)     ,
  product_monthly_sales_cnt integer ,
  product_comment_time date       ,
  product_comment_num integer     ,
  product_comment_content varchar(200)
)
with (
  orientation = column,
  compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Run the **INSERT INTO .. SELECT ..** command to import data from the foreign table to the target table.

Example:

```
INSERT INTO product_info SELECT * FROM foreign_product_info;
```

If information similar to the following is displayed, the data has been imported.

```
INSERT 0 20
```

Step 3 Run the following **SELECT** command to view data imported from MRS to GaussDB(DWS):

```
SELECT * FROM product_info;
```

If the query result displays data specified in [Data File](#), the import is successful. The following information is displayed at the end of the query result:

(20 rows)

----End

7.7.6 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If [Querying Data After Importing It](#) is performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

Step 2 Run the following command to delete the foreign table:

```
DROP FOREIGN TABLE foreign_product_info;
```

----End

Deleting the Manually Created Foreign Server

If [Manually Creating a Foreign Server](#) is performed, perform the following steps to delete the foreign server, database, and user:

Step 1 Use the client provided by GaussDB(DWS) to connect to the database where the foreign server resides as the user who created the foreign server.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

```
\c mydatabase dbuser;
```

Enter the password as prompted.

- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:

```
gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
```

Enter the password as prompted.

Step 2 Delete the manually created foreign server.

Run the following command to delete the server. For details about the syntax, see **DROP SERVER**.

```
DROP SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca;
```

The foreign server is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE  
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

The server is successfully deleted if the returned result is as follows:

```
srvname | srvowner | srvfwdw | srvtpe | srversion | srvacl | srvoptions  
-----+-----+-----+-----+-----+-----+-----  
(0 rows)
```

Step 3 Delete the customized database.

Connect to the default database **postgres** through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:

```
DROP DATABASE
```

Step 4 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:

```
REVOKE ALL ON FOREIGN DATA WRAPPER hdfs_fdw FROM dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

7.7.7 Error Handling

The following error information indicates that GaussDB(DWS) is to read an ORC data file but the actual file is in TXT format. Therefore, create a table of the Hive ORC type and store the data to the table.

```
ERROR: dn_6009_6010: Error occurs while creating an orc reader for file /user/hive/warehouse/products_info.txt, detail can be found in dn log of dn_6009_6010.
```

7.8 Importing Stream Data from DIS to GaussDB(DWS)

You can use Data Ingestion Service (DIS) to import data from DIS to the database of a data warehouse cluster in real time. Stream data stored in DIS streams will be periodically imported to GaussDB(DWS). Before data is imported to

GaussDB(DWS), it is stored on OBS as temporary data and will be deleted after it is dumped to GaussDB(DWS).

Copying data from DIS to GaussDB(DWS) involves three phases:

1. [Creating a Data Warehouse Cluster, Database, and Table](#)
2. [Creating a DIS Stream and Accessing Real-Time Data](#)
3. [Viewing Data Imported from DIS](#)

Creating a Data Warehouse Cluster, Database, and Table

Step 1 Create a data warehouse cluster.

For details, see [Creating Clusters](#).

If there is one available, skip this step.

For example, create a data warehouse cluster named **dws-demo**.

Step 2 Use an SQL client to access the data warehouse cluster.

For details, see [Methods of Connecting to a Cluster](#).

You can choose any method to connect to a cluster.

Step 3 On the SQL client, run the SQL statements to create a database and a GaussDB(DWS) table, and configure the schema.

For details, see [Before You Start](#).

- Create a database account whose username is **joe** and password is **Bigdata@123**.
- Set the database name to **db_tpcds**.
- Set the schema to **myschema**.

If you do not specify the schema, **public** is adopted by default.

- Create a GaussDB(DWS) table named **mytable**.

When creating the GaussDB(DWS) table, design its structure based on the source data. Fields and field types in the GaussDB(DWS) table must correspond to their sources.

----End

Creating a DIS Stream and Accessing Real-Time Data

Step 1 Log in to the DIS management console and create a DIS stream.

For details, see [Creating a DIS Stream](#).

The DIS streams for importing data from DIS to GaussDB(DWS) have the following requirements:

- **Region:** Select the same region that the data warehouse cluster resides. Currently, only **CN North-Beijing4** is supported.
- **Source Data Type:** Only **CSV** is supported.

Step 2 On the DIS console, add a dump task for the newly purchased stream. Set **Dump Destination** to **GaussDB(DWS)** to dump stream data to GaussDB(DWS).

For details, see [Creating a Dump Task](#).

When adding a dump task, set GaussDB(DWS) parameters according to [Creating a Data Warehouse Cluster, Database, and Table](#). The parameters are as follows:

- **Dump Destination:** Select **GaussDB(DWS)**. Stream data is stored in DIS, and periodically imported to GaussDB(DWS). Before data is imported to GaussDB(DWS), it is stored on OBS as temporary data and will be deleted after it is dumped to GaussDB(DWS).
- **GaussDB(DWS) Cluster:** Name of the data warehouse cluster to which the data will be dumped, for example, **dws-demo**
- **GaussDB(DWS) Database:** Name of the GaussDB(DWS) database to which the dumped data will be stored, for example, **db_tpcds**
- **Database Schema:** Schema of the GaussDB(DWS) database, for example, **myschema**
- **GaussDB(DWS) Table:** GaussDB(DWS) tables in the specified database schema, for example, **mytable**
- **Delimiter:** Delimiter used to separate row data generated when data is dumped to a GaussDB(DWS) data table
- **Username:** Username of the target GaussDB(DWS) database for data dumping. The database user must have the read and write permissions on **GaussDB(DWS) Table**, for example, **joe**.
- **Password:** Password of the user specified by **Username**.

Step 3 Set up the environment for creating a DIS application and send real-time data to DIS.

For details, see **Getting Started > Step 2: Preparing a DIS Application Development Environment** and **Getting Started > Step 3: Sending Data to DIS** in the *Data Ingestion Service User Guide*.

----End

Viewing Data Imported from DIS

Step 1 On the SQL client, connect to the database storing the imported data in the data warehouse cluster.

For details, see [Methods of Connecting to a Cluster](#).

You can choose any method to connect to a cluster.

Step 2 Run the query command to view the data.

The following is an example command. Replace *table_name* to the value of **GaussDB(DWS) Table** you set when creating the stream.

```
SELECT * FROM table_name;
```

----End

7.9 Using CDM to Migrate Data to GaussDB(DWS)

You can use CDM to migrate data from other data sources (for example, MySQL) to the databases in clusters on GaussDB(DWS).

For details about scenarios where CDM is used to migrate data to GaussDB(DWS), see the following sections of *Cloud Data Migration User Guide*.

- **Getting Started**: describes how to use CDM to migrate local MySQL databases to GaussDB(DWS).
- **From DDS to GaussDB(DWS)**

7.10 Using the DSC Tool to Migrate SQL Scripts

The Database Schema Converter (DSC) is a command-line tool running on the Linux or Windows OS. It is dedicated to providing customers with simple, fast, reliable application SQL script migration services. It parses SQL scripts of source database applications by using the built-in syntax migration logic, and migrates them to be applicable to GaussDB(DWS) databases. You do not need to connect the DSC to a database. It can migrate data in offline mode without service interruption. In GaussDB(DWS), you can run the migrated SQL scripts to restore the database, thereby easily migrating offline databases to the cloud.

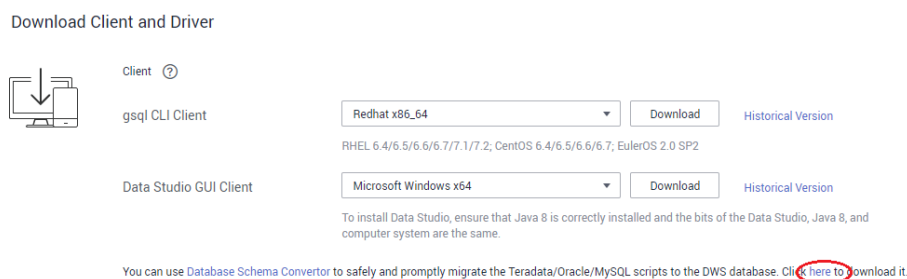
The DSC can migrate SQL scripts of Teradata, Oracle, Netezza, MySQL, and DB2 databases.

Downloading the DSC SQL Migration Tool

- Step 1** Log in to the GaussDB(DWS) management console.
- Step 2** In the navigation tree on the left, click **Connection Management**.
- Step 3** In the **Download Client and Driver** area, click **here** to download the DSC migration tool.

If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the client corresponding to the cluster version. In the cluster list on the **Cluster Management** page, click the name of the specified cluster and click the **Basic Information** tab to view the cluster version.

Figure 7-8 Downloading the tool



----End

Operation Guide for the DSC SQL Migration Tool

For details, see [DSC - SQL Syntax Migration Tool](#).

7.11 Checking for Data Skew

Scenarios

Data skew causes the query performance to deteriorate. Before importing all the data from a table consisting of over 10 million records, you are advised to import some of the data and check whether data skew occurs and whether the distribution keys need to be changed. Troubleshoot the problems if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Context

GaussDB(DWS) uses a massively parallel processing (MPP) system of the shared-nothing architecture. The MPP performs horizontal partitioning to store tuples in the service data table on all DN's using proper distribution policies.

The following user table distribution policies are supported:

- Replication: stores a full table on each DN. You are advised to use the replication mode for tables containing a small volume of data.
- Hash: A distribution key must be specified for a user table. When a record is inserted, the system hashes it based on the distribution key and then stores it on the corresponding DN. You are advised to use the hash distribution policy for tables with a large volume of data.

If an inappropriate distribution key is used, data skew may occur when you use the hash policy. Check for data skew when you use the hash distribution policy so that data can be evenly distributed to each DN. You are advised to use the column with few replicated values as the distribution key.

Procedure

Step 1 Analyze the source data and select candidate distribution keys.

Step 2 Select a column from the candidates in [Step 1](#) as the distribution key to create a target table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
| table_constraint | LIKE source_table [ like_option [ ... ] ] }
[, ... ]) [ WITH ( {storage_parameter = value} [, ... ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ COMPRESS | NOCOMPRESS ] [ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION
| { HASH ( column_name [, ... ] ) } } ];
```

Step 3 Import a small batch of data to the target table.

When importing a single data file, you can evenly split this file and import a part of it to check for the data skew in the target table.

Step 4 Check for data skew. (Replace *table_name* with the actual table name.)

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP
BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

Step 5 If the data distribution deviation is less than 10% across DNs, data is evenly distributed and an appropriate distribution key has been selected. Clear the imported small-batch data and import full data to complete data migration.

If data distribution deviation across DNs is greater than or equal to 10%, data skew occurs. Remove this distribution key from the candidates in [Step 1](#), delete the target table, and repeat [Step 2](#) through [Step 5](#).

Step 6 (Optional) If you fail to select an appropriate distribution key after performing the preceding steps, select multiple columns from the candidates as distribution keys.

----End

Examples

Assume you want to select an appropriate distribution key for the **staffs** table.

1. Analyze the source data for the **staffs** table and select the **staff_ID**, **FIRST_NAME**, and **LAST_NAME** columns as candidate distribution keys.
2. Select the **staff_ID** column as the distribution key and create the target table **staffs**.

```
CREATE TABLE staffs
(
  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(2,2),
  MANAGER_ID    NUMBER(6),
  section_ID    NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID);
```

3. Import a small batch of data to the target table **staffs**.

There are eight DNs in the cluster based on the following query, and you are advised to import 80,000 records.

```
SELECT count(*) FROM pgxc_node where node_type='D';
count
-----
      8
(1 row)
```

4. Verify the data skew of the target table **staffs** whose distribution key is **staff_ID**:

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
-----+-----
11010 | datanode4
10000 | datanode3
12001 | datanode2
 8995 | datanode1
10000 | datanode5
 7999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```


- The preceding query result indicates that the distribution deviation across DNs is greater than 10%. The data skew occurs. Therefore, delete **staff_ID** from the distribution key candidates and delete the **staffs** table.

```
DROP TABLE staffs;
```

- Use **staff_ID**, **FIRST_NAME**, and **LAST_NAME** as distribution keys and create the target table **staffs**.

```
CREATE TABLE staffs
(
  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(2,2),
  MANAGER_ID    NUMBER(6),
  section_ID    NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID,FIRST_NAME,LAST_NAME);
```

- Verify the data skew of the target table **staffs** whose distribution keys are **staff_ID**, **FIRST_NAME**, and **LAST_NAME**.

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

```
count | node_name
-----+-----
10010 | datanode4
10000 | datanode3
10001 | datanode2
9995  | datanode1
10000 | datanode5
9999  | datanode6
9995  | datanode7
10000 | datanode8
(8 rows)
```

- The preceding query result indicates that the data deviation across DNs is less than 10%. The data is evenly distributed and the appropriate distribution keys have been selected.
- Delete the imported small-batch data.

```
TRUNCATE TABLE staffs;
```
- Import the full data to complete data migration.

7.12 Analyzing a Table

The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the **ANALYZE** statement to update table statistics. The statistics are stored in the **PG_STATISTIC** system catalog.

Analyzing a Table

ANALYZE supports row-store, column-store, HDFS, and OBS tables in ORC or CARBONDATA format. **ANALYZE** can also collect statistics about specified columns of a local table. For details about **ANALYZE**, see [ANALYZE](#) | [ANALYZE](#).

Step 1 Update table statistics.

Do **ANALYZE** to the **product_info** table.

```
ANALYZE product_info;  
ANALYZE
```

----End

Automatically Analyzing a Table

GaussDB(DWS) provides the GUC parameter `autovacuum` to specify whether to enable the autovacuum function of the database.

If **autovacuum** is set to **on**, the system will start the autovacuum thread to automatically analyze tables when the data volume in the table reaches the threshold. This is the autoanalyze function.

- For an empty table, when the number of rows inserted to it is greater than 50, **ANALYZE** is automatically triggered.
- For a table containing data, the threshold is $50 + 10\% \times \text{reltuples}$, where **reltuples** indicates the total number of rows in the table.

The autovacuum function also depends on the following two GUC parameters in addition to **autovacuum**:

- **track_counts**: This parameter must be set to **on** to enable statistics collection about the database.
- `autovacuum_max_workers`: This parameter must be set to a value greater than 0 to specify the maximum number of concurrent autovacuum threads.

NOTICE

- The autoanalyze function supports only the default sampling mode and not the percentage sampling mode.
 - The autoanalyze function does not collect multi-column statistics, which only supports percentage sampling.
 - The autoanalyze function supports row-store and column-store tables and does not support the following: foreign tables, HDFS tables, OBS tables, temporary tables, unlogged tables, and toast tables.
-

7.13 Doing VACUUM to a Table

If a large number of rows were updated or deleted during import, run **VACUUM FULL** before **ANALYZE**. A large number of updates and delete operations generate huge disk page fragments, which reduces the query efficiency. **VACUUM FULL** can restore disk page fragments and return them to the OS.

Step 1 Run the **VACUUM FULL** statement.

Do **VACUUM FULL** to the **product_info** table:

```
VACUUM FULL product_info  
VACUUM
```

----End

7.14 Managing Concurrent Write Operations

7.14.1 Transaction Isolation

GaussDB(DWS) manages transactions based on MVCC and two-phase locks, avoiding conflicts between read and write operations. SELECT is a read-only operation, whereas UPDATE and DELETE are read/write operations.

- There is no conflict between read/write and read-only operations, or between read/write operations. Each concurrent transaction creates a snapshot when it starts. Concurrent transactions cannot detect updates made by each other.
 - At the **READ COMMITTED** isolation level, if transaction T1 is committed, transaction T2 can see changes made by T1.
 - At the **REPEATABLE READ** level, if T2 starts before T1 is committed, T2 will not see changes made by T1 even after T1 is committed. The query results in a transaction are consistent and unaffected by other transactions.
- Read/Write operations use row-level locks. Different transactions can concurrently update the same table but not the same row. A row update transaction will start only after the previous one is committed.
 - **READ COMMITTED**: At this level, a transaction can access only committed data. This is the default level.
 - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.

7.14.2 Write and Read/Write Operations

Statements for write-only and read/write operations are as follows:

- **INSERT**, used to insert one or more rows of data into a table.
- **UPDATE**, used to modify existing data in a table.
- **DELETE**, used to delete existing data from a table.
- **COPY**, used to import data.

INSERT and COPY are write-only operations. Only one of them can be performed at a time. If the INSERT or COPY of transaction T1 locks a table, the INSERT or COPY of transaction T2 needs to wait until T1 unlocks the table.

UPDATE and DELETE operations are read/write operations. They need to query for the target rows before modifying data. Concurrent transactions cannot see the changes made by each other, and UPDATE and DELETE operations read snapshots of data committed before their transactions started. Write operations use row-level locks. If T2 starts after T1 and is to update the same row as T1 does, T2 waits for T1 to finish update. If T1 is not complete within the specified timeout duration, T2 will time out. If T1 and T2 updates different rows in a table, they can be concurrently executed.

7.14.3 Potential Deadlocks During Concurrent Write

Whenever transactions involve updates of more than one table, there is always the possibility of concurrently running transactions becoming deadlocked when they both try to write to the same set of tables. A transaction releases all of its locks at once when it either commits or rolls back; it does not relinquish locks one at a time. For example, suppose that transactions T1 and T2 start at roughly the same time.

- If T1 starts writing to table A and T2 starts writing to table B, both transactions can proceed without conflict; however, if T1 finishes writing to table A and needs to start writing to the same rows as T2 does in table B, it will not be able to proceed because T2 still holds the lock on B. Conversely, if T2 finishes writing to table B and needs to start writing to the same rows as T1 does in table A, it will not be able to proceed either because T1 still holds the lock on A. In this case, a deadlock occurs. If T1 is committed and releases the lock within the lock timeout duration, subsequent update will proceed. If a lock times out, an error will be reported and the corresponding transaction will exit.
- If T1 updates rows 1 to 5 and T2 updates rows 6 to 10 in the same table, the two transactions do not conflict. However, if T1 finishes the update and proceeds to update rows 6 to 10, and T2 proceeds to update rows 1 to 5, neither of them can continue. If either of the transactions is committed and releases the lock within the lock timeout duration, subsequent update will proceed. If a lock times out, an error will be reported and the corresponding transaction will exit.

7.14.4 Concurrent INSERT and DELETE in the Same Table

Transaction T1:

```
START TRANSACTION;  
INSERT INTO test VALUES(1,'test1','test123');  
COMMIT;
```

Transaction T2:

```
START TRANSACTION;  
DELETE test WHERE NAME='test1';  
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, the DELETE of T2 is performed. In this case, **DELETE 0** is displayed, because T1 is not committed and T2 cannot see the data inserted by T1.

Scenario 2:

- **READ COMMITTED** level
T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed and the DELETE of T2 is performed. In this case, **DELETE 1** is displayed, because T2 can see the data inserted by T1.
- **REPEATABLE READ** level
T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed and the DELETE of T2 is

performed. In this case, **DELETE 0** is displayed, because the data obtained in queries is consistent in a transaction.

7.14.5 Concurrent INSERT in the Same table

Transaction T1:

```
START TRANSACTION;  
INSERT INTO test VALUES(2,'test2','test123');  
COMMIT;
```

Transaction T2:

```
START TRANSACTION;  
INSERT INTO test VALUES(3,'test3','test123');  
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, the INSERT of T2 is performed and succeeds. At the **READ COMMITTED** and **REPEATABLE READ** levels, the SELECT of T1 cannot see data inserted by T2, and a query in T2 cannot see the data inserted by T1.

Scenario 2:

- **READ COMMITTED** level

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed. In T2, a query performed after INSERT can see the data inserted by T1.

- **REPEATABLE READ** level

T1 is started but not committed. At the same time, T2 is started. After the INSERT of T1 is complete, T1 is committed. In T2, a query performed after INSERT cannot see the data inserted by T1.

7.14.6 Concurrent UPDATE in the Same Table

Transaction T1:

```
START TRANSACTION;  
UPDATE test SET address='test1234' WHERE name='test1';  
COMMIT;
```

Transaction T2:

```
START TRANSACTION;  
UPDATE test SET address='test1234' WHERE name='test2';  
COMMIT;
```

Transaction T3:

```
START TRANSACTION;  
UPDATE test SET address='test1234' WHERE name='test1';  
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. The UPDATE of T1 and then T2 starts, and both of them succeed. This is because UPDATE operations use row-level locks and do not conflict when they update different rows.

Scenario 2:

T1 is started but not committed. At the same time, T3 is started. The UPDATE of T1 and then T3 starts, and the UPDATE of T1 succeeds. The UPDATE of T3 times out. This is because T1 and T3 update the same row, the lock on which is held by the uncommitted T1.

7.14.7 Concurrent Data Import and Queries

Transaction T1:

```
START TRANSACTION;  
COPY test FROM '...';  
COMMIT;
```

Transaction T2:

```
START TRANSACTION;  
SELECT * FROM test;  
COMMIT;
```

Scenario 1:

T1 is started but not committed. At the same time, T2 is started. The COPY of T1 and then the SELECT of T2 starts, and both of them succeed. In this case, T2 cannot see the data added by the COPY of T1.

Scenario 2:

- **READ COMMITTED** level

T1 is started but not committed. At the same time, T2 is started. The COPY of T1 is complete and T1 is committed. In this case, T2 can see the data added by the COPY of T1.

- **REPEATABLE READ** level

T1 is started but not committed. At the same time, T2 is started. The COPY of T1 is complete and T1 is committed. In this case, T2 cannot see the data added by the COPY of T1.

8 Data Export

8.1 Exporting Data to OBS in Parallel

NOTICE

- Ensure no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the data warehouse cluster are in the same region.

8.1.1 Parallel OBS Data Export

Overview

GaussDB(DWS) databases allow you to export data in parallel using OBS foreign tables, in which the export mode and the exported data format are specified. Data is exported in parallel through multiple DNs from GaussDB(DWS) to the OBS server, improving the overall export performance.

- A CN only plans and delivers data export tasks, and DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.
- You can concurrently export data using multiple OBS services, but the bucket and object paths specified for the export tasks must be different and not null.
- The selected OBS server should connect to cluster nodes. Its export rate is affected by network bandwidths.
- The TEXT and CSV data file formats are supported. The size of data in a single row must be less than 1 GB.

Related Concepts

- **Source data file:** a TEXT or CSV file that stores data.

- **OBS:** a cloud storage service used to store unstructured data, such as documents, images, and videos. Data objects concurrently exported from GaussDB(DWS) are stored on the OBS server.
- **Bucket:** a container storing objects on OBS.
 - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
 - In OBS, each bucket name must be unique and cannot be changed. A default ACL is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.
 - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object:** a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.

Generally, objects are managed as files. However, OBS has no file system-related concepts, such as files and folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key:** name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name+Object name*.
- **Metadata:** object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
 - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
 - User metadata contains object descriptions specified by users for uploading objects.
- **Data:** object content. OBS does not sense the content and regards it as stateless binary data.
- **Foreign table:** A foreign table is used to identify data in a source data file. It stores information, such as the location, format, destination location, encoding format, and data delimiter of a source data file.

Principles

The following describes the principles of exporting data from a cluster to OBS by using a distributed Hash table or a replication table.

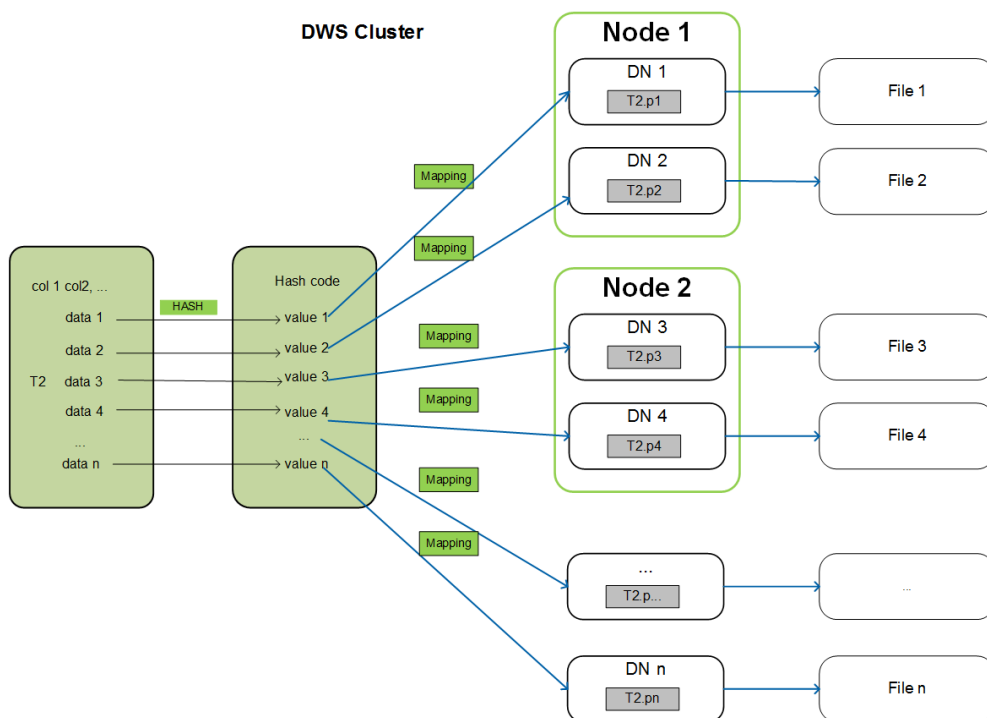
- **Distributed Hash table:** the table for which **DISTRIBUTE BY HASH (Column_Name)** is specified in the table creation statement.

A distributed Hash table stores data in Hash mode. [Figure 8-1](#) shows how to export data from table (T2) to OBS as an example.

During table data storage, the **col2** Hash field in table **T2** is hashed, and a Hash value is generated. The tuple is distributed to corresponding DNs for storage according to the mapping between the DNs and the Hash value.

When data is exported to OBS, DNs that store the exported data of **T2** directly export their data files to OBS. Original data on multiple nodes will be exported in parallel.

Figure 8-1 Hash distribution principle



- Replication table: the table for which **DISTRIBUTE BY REPLICATION** is specified in the table creation statement.

A replication table stores a package of complete table data on each GaussDB(DWS) node. When exporting data to OBS, GaussDB(DWS) randomly selects a DN for export.

Naming Convention of Exported Files

Rules for naming the files exported from GaussDB(DWS) to OBS are as follows:

- Data exported from DNs is stored on OBS in segment format. The file is named as *Table name_Node name_segment.n*. *n* is a natural number starting from 0, for example, 0, 1, 2, 3.

For example, the data of table **t1** on **datanode3** will be exported as **t1_datanode3_segment.0**, **t1_datanode3_segment.1**, and so on.

You are advised to export data from different clusters or databases to different OBS buckets or different paths of the same OBS bucket.

- Each segment can store a maximum of 1 GB data, with no tuples sliced. If data stored in a segment exceeds 1 GB, the excess data will be stored in the second segment.

For example:

A segment has already stored 100 pieces of tuples (1023 MB) when **datanode3** exports data from **t1** to OBS. If a 5 MB tuple is inserted to the segment, the data size becomes 1028 MB. In this case, file **t1_datanode3_segment.0** (1023 MB) is generated and stored on OBS, and the new tuple is stored on OBS as file **t1_datanode3_segment.1**.

- When data is exported from a distributed Hash table, the number of segments generated on each DN depends on the data volume stored on a DN, irrelevant to the number of DNs in the cluster. Data stored in Hash mode may not be evenly distributed on each DN.

For example, a cluster has DataNode1, DataNode2, DataNode3, DataNode4, DataNode5, and DataNode6, which store 1.5 GB, 0.7 GB, 0.6 GB, 0.8 GB, 0.4 GB, and 0.5 GB data, respectively. Seven OBS segment files will be generated during data export because DataNode1 will generate two segment files, which store 1 GB and 0.5 GB data, respectively.

Data Export Process

Figure 8-2 Concurrent data export

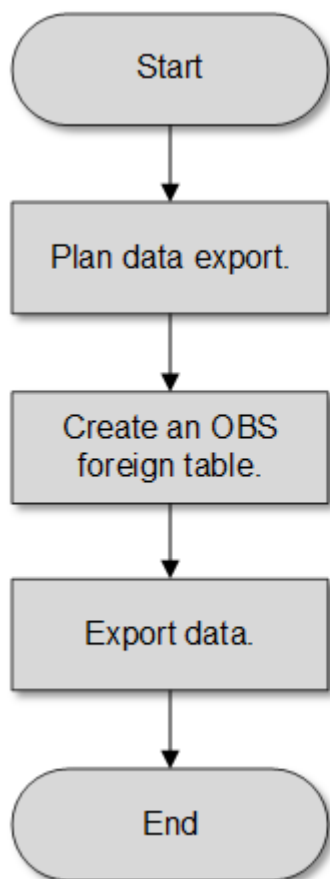


Table 8-1 Process description

Procedure	Description	Subtask
Plan data export.	Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data files. For details, see Planning Data Export .	-
Create an OBS foreign table.	Create a foreign table to help OBS specify information about data files to be exported. The foreign table stores information, such as the target location, format, encoding, and inter-data delimiter of a source data file. For details, see Creating an OBS Foreign Table .	-
Export data.	After the foreign table is created, run the INSERT statement to efficiently export data to data files. For details, see Exporting Data .	-

8.1.2 Planning Data Export

Scenarios

Plan the storage location of exported data in OBS.

Planning OBS Save Path and File

You need to specify the OBS path (to directory) for storing data that you want to export. The exported data can be saved to a file in CSV format. The system also supports TEXT so that you can import the exported data to various applications.

The target directory cannot contain any files.

Planning OBS Bucket Permissions

The user used to export data must:

- Have OBS enabled.
- Have the write permission on the OBS bucket where the data export path is located.

You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

For details, see [Granting Write Permission to OBS Storage Location and OBS Bucket as Planned](#).

Planning Data to Be Exported and Foreign Tables

You must prepare data to be exported in the database table, and the data volume per row must be less than 1 GB. Based on the data to be exported, plan foreign tables whose attributes such as columns, column types, and length match those of user data.

Granting Write Permission to OBS Storage Location and OBS Bucket as Planned

Step 1 Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data.

1. Log in to the OBS management console.
Click **Service List** and choose **Object Storage Service** to open the OBS management console.
Alternatively, you can access <https://storage.huaweicloud.com> to log in to the OBS management console.
2. Create a bucket.
For details about how to create a bucket, see [Creating a Bucket](#) in the *Object Storage Service Console Operation Guide*.
For example, create a bucket named **mybucket**.
3. Create a folder.
In the OBS bucket, create a folder for storing exported data.
For example, create a folder named **output_data** in the created **mybucket** OBS bucket.

Step 2 Determine the path of the created OBS folder.

Specify the OBS path for storing exported data files. This path is the value of the **location** parameter used for creating a foreign table.

The OBS folder path in the **location** parameter consists of **obs://**, a bucket name, and a file path. Example:

`obs://<bucket_name>/<file_path>`

In this example, the OBS folder path is as follows:

```
obs://mybucket/output_data/
```

Step 3 Grant the OBS bucket write permission to the user who wants to export data.

When exporting data, a user must have the write permission on the OBS bucket where the data export path is located. You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

----End

8.1.3 Creating an OBS Foreign Table

Procedure

- Step 1** Based on the path planned in [Planning Data Export](#), determine the value of the **location** parameter used for creating a foreign table.
- Step 2** Obtain access keys (AK and SK) for accessing OBS. To obtain access keys, log in to the management console, move the cursor over the username in the upper right corner, and click **My Credentials**. Then choose **Access Keys** in the navigation tree on the left. On the **Access Keys** page, you can view the existing AKs or click **Create Access Key** to create an AK/SK pair.
- Step 3** Examine the formats of data to be exported and determine the values of data format parameters used for creating a foreign table. For details, see data format parameters.
- Step 4** Create an OBS table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Examples

Create a foreign table in the GaussDB(DWS) database. Set parameters as follows:

- **location**
The OBS path of the source data file has been obtained in [step 2](#) in [Planning Data Export](#).

For example, set **location** as follows:

```
location 'obs://mybucket/output_data/',
```

- **Access keys (AK and SK)**
 - Set **access_key** to the AK you have obtained.
 - Set **secret_access_key** to the SK you have obtained.

NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**
 - Set **format** to **CSV**.
 - Set **encoding** to **UTF-8**.
 - Set **encrypt** to **off**.
 - Set **delimiter** to **,**.
 - Set **header** (whether the exported data file contains the header row).
Specifies whether a file contains a header with the names of each column in the file.
When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**, indicating that the first row of the exported data file is not the header.

Based on the preceding settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext;  
CREATE FOREIGN TABLE product_info_output_ext  
(  
    product_price          integer      not null,  
    product_id             char(30)     not null,  
    product_time           date          ,  
    product_level          char(10)     ,  
    product_name           varchar(200) ,  
    product_type1          varchar(20)  ,  
    product_type2          char(10)     ,  
    product_monthly_sales_cnt integer    ,  
    product_comment_time   date          ,  
    product_comment_num    integer      ,  
    product_comment_content varchar(200)  
)  
SERVER gsmpp_server  
OPTIONS(  
    location 'obs://mybucket/output_data/',  
    FORMAT 'CSV' ,  
    DELIMITER ';;',  
    encoding 'utf8',  
    header 'false',  
    ACCESS_KEY 'access_key_value_to_be_replaced',  
    SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'  
)  
WRITE ONLY ;
```

If the following information is displayed, the foreign table has been created:

```
CREATE FOREIGN TABLE
```

8.1.4 Exporting Data

Procedure

Step 1 Export data.

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

NOTE

- Only a single internal table can be exported at a time. Multi-table join is not supported during export. Results of the aggregation, sort, subquery, and limit operations on a single table also cannot be exported.

----End

Examples

- Example 1:** Export data from table **product_info_output** to a data file through the **product_info_output_ext** foreign table.

```
DWSINSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

If information similar to the following is displayed, the data has been exported.

```
INSERT 0 10
```

- Example 2:** Export part of the data to a data file by specifying the filter condition **WHERE product_price>500**.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
```

 NOTE

Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to the hexadecimal format before export.

8.1.5 Examples

Exporting a Table

Create two foreign tables and use them to export tables from a database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input_data1** and **/input_data2** for storing data files, and create data directories **/input_data1/data** and **/input_data2/data**, respectively, in the two buckets.

Step 2 On the GaussDB(DWS) database, create the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS data server to receive data exported from the database.

OBS and the database are in the same region. The example GaussDB(DWS) table to be exported is **tpcds.customer_address**.

Export information is set as follows:

- The source data file directories are **/input_data1/data/** and **input_data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input_data1/data/** and **obs://input_data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from a database. The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **0E08**.
- **quote** is set to **0x1b**.
- **encrypt** is set to **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

 NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
  ca_address_sk          integer          ,
  ca_address_id          char(16)         ,
  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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
);
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk        integer
ca_address_id        char(16)
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
);
```

Step 3 In GaussDB(DWS), export the data table **tpcds.customer_address** to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** concurrently.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address;
```

NOTE

The design of OBS foreign tables does not allow exporting files to a non-empty path. However, in concurrent export scenarios, multiple files are exported to the same path, causing an error.

Assume that a user concurrently exports data from the same table to the same OBS foreign table, and that one SQL statement is executed to export data when another SQL statement is being executed and has not generated any file on the OBS server. In this case, certain data is overwritten although both SQL statements are successfully executed. Therefore, you are advised not to concurrently export data to the same OBS foreign table.

----End

Concurrently Exporting Tables

Use the two foreign tables to export tables from the database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input_data1** and **/input_data2** for storing data files, and create data directories **/input_data1/data** and **/input_data2/data**, respectively, in the two buckets.

Step 2 In GaussDB(DWS), create foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS server to receive exported data.

OBS and the database are in the same region. Tables to be exported are **tpcds.customer_address1** and **tpcds.customer_address2**.

Export information is set as follows:

- The source data file directories are **/input_data1/data/** and **input_data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input_data1/data/** and **obs://input_data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from GaussDB(DWS). The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **0E08**.
- **quote** is set to **0x1b**.
- **encrypt** is set to **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
  ca_address_sk          integer          ,
  ca_address_id          char(16)         ,
  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)
)
SERVER gsmpp_server
```

```
OPTIONS(LOCATION 'obs://input_data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
);
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk          integer          ,
ca_address_id          char(16)          ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
);
```

Step 3 In GaussDB(DWS), export the data tables **tpcds.customer_address1** and **tpcds.customer_address2** in parallel to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2**, respectively.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address1;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address2;
```

----End

8.2 Using GDS to Export Data to a Remote Server

8.2.1 Exporting Data In Parallel Using GDS

In high-concurrency scenarios, you can use GDS to export data from a database to a common file system.

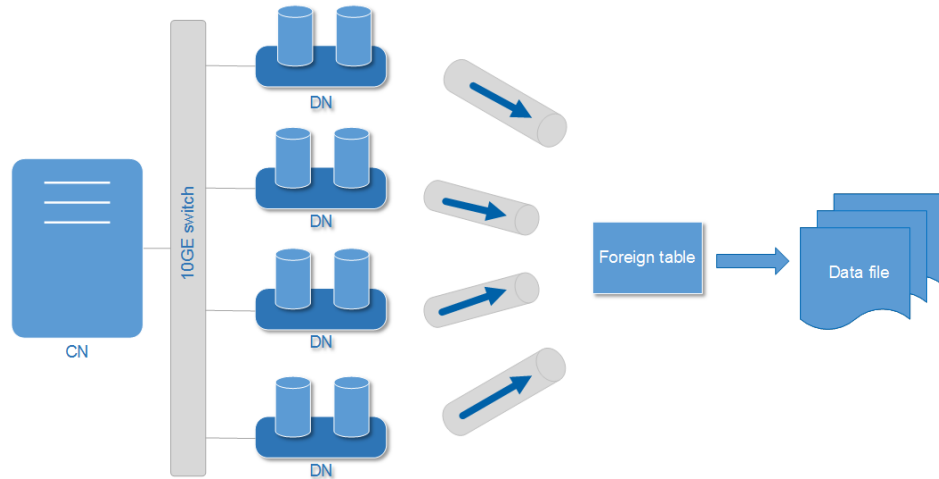
Overview

Using foreign tables: A GDS foreign table specifies the exported file format and export mode. Data is exported in parallel through multiple DNs from the database to data files, which improves the overall data export performance. The data files cannot be directly exported to HDFS.

- The CN only plans data export tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.

- In this way, the computing capacities and bandwidths of all the DN nodes are fully leveraged to export data.

Figure 8-3 Exporting data using foreign tables



Related Concepts

- **Data file:** a TEXT, CSV, or FIXED file that stores data exported from GaussDB(DWS).
- **Foreign table:** a table that stores information, such as the format, location, and encoding format of a data file.
- **GDS:** data service tool. To export data, deploy it on the server where data files are stored.
- **Table:** tables in the database, including row-store tables and column-store tables from which data is exported.
- **Remote mode:** Service data in a cluster is exported to hosts outside the cluster.

Export Modes

Data can be exported to GaussDB(DWS) in **Remote** mode.

- **Remote mode:** Service data in a cluster is exported to hosts outside the cluster.
 - In this mode, multiple GDSs are used to concurrently export data. One GDS can export data for only one cluster at a time.
 - The data export rate of a GDS that resides on the same intranet as cluster nodes is limited by the network bandwidth. A 10GE configuration is recommended.
 - Data files in TEXT or CSV format are supported. The size of data in a single row must be less than 1 GB.

Data Export Process

Figure 8-4 Process of parallel data export

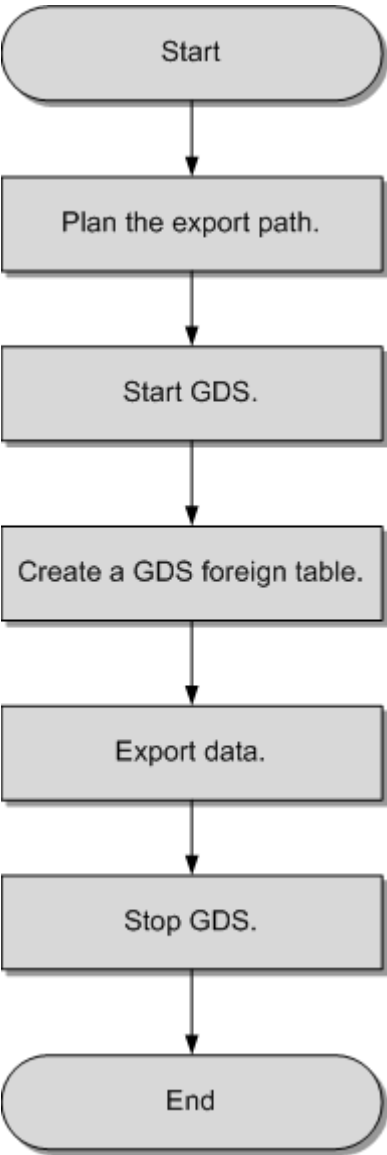


Table 8-2 Process description

Procedu re	Description	Subtask
Plan data export.	Prepare data to be exported and plan the export path for the mode to be selected. For details, see Planning Data Export .	-

Procedure	Description	Subtask
Start GDS.	If the Remote mode is selected, install, configure, and start GDS on data servers. For details, see Installing, Configuring, and Starting GDS .	-
Create a foreign table.	Create a foreign table to help GDS specify information about a data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a data file. For details, see Creating a GDS Foreign Table .	-
Export data.	After the foreign table is created, run the INSERT statement to efficiently export data to data files. For details, see Exporting Data .	-
Stop GDS.	Stop GDS after data is exported. For details, see Stopping GDS .	-

8.2.2 Planning Data Export

Scenarios

Before you use GDS to export data from a cluster, prepare data to be exported and plan the export path.

Planning an Export Path

- **Remote** mode

Step 1 Log in to the GDS data server as user **root** and create the **/output_data** directory for storing data files.

```
mkdir -p /output_data
```

Step 2 (Optional) Create a user and the user group to which it belongs. This user is used to start GDS and must have the write permission on the directory for storing data files.

```
groupadd gdsgrp  
useradd -g gdsgrp gdsuser
```

If the following information is displayed, the user and user group already exist. Skip this step.

```
useradd: Account 'gdsuser' already exists.  
groupadd: Group 'gdsgrp' already exists.
```

Step 3 Change the directory owner to **gdsuser**.

```
chown -R gdsuser:gdsgrp /output_data
```

----End

8.2.3 Installing, Configuring, and Starting GDS

GDS is a data service tool provided by GaussDB(DWS). Using the foreign table mechanism, this tool helps export data at a high speed.

For details, see [Installing, Configuring, and Starting GDS](#).

8.2.4 Creating a GDS Foreign Table

Procedure

Step 1 Set the **location** parameter for the foreign table based on the path planned in [Planning Data Export](#).

- **Remote mode**

Set the **location** parameter to the URL of the directory that stores the data files.

- You do not need to specify a file name in the URL.
- If multiple URLs are configured, only the first URL is effective.

For example:

The IP address of the GDS data server is 192.168.0.90. The listening port number set during GDS startup is 5000. The directory for storing data files is **/output_data**.

In this case, set the **location** parameter to **gsfs://192.168.0.90:5000/**.

 NOTE

location can be set to a subdirectory, for example, **gsfs://192.168.0.90:5000/2019/11/**, so that the same table can be exported to different directories by date.

In the current version, when an **export** task is executed, the system checks whether the **/output_data/2019/11** directory exists. If the directory does not exist, the system creates it. During the export, files are written to this directory. In this way, you do not need to manually run the **mkdir -p /output_data/2019/11** command after creating or modifying a foreign table.

Step 2 Set data format parameters in the foreign table based on the planned data file formats. For details about format parameters, see data format parameters in CREATE FOREIGN TABLE (for GDS Import and Export).

Step 3 Create a GDS foreign table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Examples

- **Example :** Create the GDS foreign table **foreign_tpcds_reasons** for the source data. Data is to be exported as CSV files.

Data export mode settings are as follows:

The data server resides on the same intranet as the cluster. The IP address of the data server is 192.168.0.90. Data is to be exported as CSV files. The **Remote** mode is selected for parallel data export.

Assume that the directory for storing data files is **/output_data/** and the GDS listening port is 5000 when GDS is started. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **0x1b**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to a double quotation mark.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.
- **EOL** is set to **0X0A**.

The foreign table is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL '',
EOL '0x0a'
)
WRITE ONLY;
```

8.2.5 Exporting Data

Prerequisites

Ensure that the IP addresses and ports of servers where CNs and DN are deployed can connect to those of the GDS server.

Procedure

Step 1 Export data.

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

 NOTE

- Create batch processing scripts to export data in parallel. The degree of parallelism depends on the server resource usage. You can test several tables and monitor resource usage to determine whether to increase or reduce the amount. Common resource monitoring commands include **top** for monitoring memory and CPU usage, **iostat** for monitoring I/O usage, and **sar** for monitoring networks. Examples are provided in [Example: Exporting Data Using Multiple Threads](#).
- Only a single internal table can be exported at a time. Multi-table join is not supported during export. Results of the aggregation, sort, subquery, and limit operations on a single table also cannot be exported.

----End

Examples

- **Example 1:** Export data from the **reasons** table to data files through the **foreign_tpcds_reasons** foreign table.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM reasons;
```
- **Example 2:** Export part of the data to data files by specifying the filter condition **r_reason_sk=1**.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM reasons WHERE r_reason_sk=1;
```
- **Example 3:** Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the **RAWTOHEX()** function to convert it to hexadecimal the format before export.

```
INSERT INTO foreign_blob_type_tab SELECT RAWTOHEX(c) FROM blob_type_tab;
```

8.2.6 Stopping GDS

GDS is a data service tool provided by GaussDB(DWS). Using the foreign table mechanism, this tool helps export data at a high speed.

For details, see [Stopping GDS](#).

8.2.7 Example

Example: Exporting Data in Remote Mode

A data server resides on the same intranet as the cluster, the IP address of the data server is 192.168.0.90, and data files are in CSV format. In this scenario, data is concurrently exported in **Remote** mode.

To concurrently export data in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create user **gds_user** and its user group.

```
mkdir -p /output_data
```
2. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.

```
groupadd gdsgrp  
useradd -g gdsgrp gds_user
```
3. Change the owner of the **/output_data** directory on the data server to **gds_user**.

```
chown -R gds_user:gdsgrp /output_data
```
4. Log in to the data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/dws/gds**. The directory for storing data files is **/output_data/**. The IP address of the data server is 192.168.0.90. The GDS listening port is 5000. GDS runs in daemon mode.

```
/opt/bin/dws/gds/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

5. In the database, create the foreign table **foreign_tpcds_reasons** for receiving data from the data server.

Data export mode settings are as follows:

- The directory for storing exported files is **/output_data/** and the GDS monitoring port number is 5000 while GDS is started. The export directory created above is also **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **0x1b**.
- **null** is set to an empty string without quotation marks.
- **escape** is the same as **quote** by default.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/', FORMAT 'CSV',ENCODING 'utf8',DELIMITER E'\x08', QUOTE E'\x1b', NULL '') WRITE ONLY;
```

6. In the database, export data to data files through the foreign table **foreign_tpcds_reasons**.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM reasons;
```
7. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is 128954.

```
ps -ef|grep gds
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
kill -9 128954
```

Example: Exporting Data Using Multiple Threads

As planned, a GDS data server resides on the same intranet as a cluster, the IP address of the data server is 192.168.0.90, and data source files are in CSV format. In this scenario, data is concurrently exported to two target tables using multiple threads in **Remote** mode.

To concurrently export data using multiple threads in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create the database user and its user group.

```
mkdir -p /output_data
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

2. Change the owner of the **/output_data** directory on the data server to **gds_user**.

```
chown -R gds_user:gdsgrp /output_data
```

3. Log in to the data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/dws/gds**. The directory for storing data files is **/output_data/**. The IP address of the data server is 192.168.0.90. The GDS listening port is 5000. GDS runs in daemon mode. The degree of parallelism is 2.

```
/opt/bin/dws/gds/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2
```

4. In GaussDB(DWS), create the foreign tables **foreign_tpcds_reasons1** and **foreign_tpcds_reasons2** for receiving data from the data server.

- Data export mode settings are as follows:

- The directory for storing exported files is **/output_data/** and the GDS monitoring port number is 5000 while GDS is started. The export directory created above is also **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

- Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **0x1b**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to a double quotation mark.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

Based on the preceding settings, the foreign table **foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/', FORMAT 'CSV', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '') WRITE ONLY;
```

Based on the above settings, the foreign table **foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/', FORMAT 'CSV', DELIMITER E'\x08', QUOTE E'\x1b', NULL '') WRITE ONLY;
```

5. In the database, export data from table **reasons1** through the foreign table **foreign_tpcds_reasons1** and from table **reasons2** through the foreign table **foreign_tpcds_reasons2** to **/output_data**.

```
INSERT INTO foreign_tpcds_reasons1 SELECT * FROM reasons1;  
INSERT INTO foreign_tpcds_reasons2 SELECT * FROM reasons2;
```

6. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is 128954.

```
ps -ef|grep gds  
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D -t 2  
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds  
kill -9 128954
```

8.3 Using gs_dump and gs_dumpall to Export Data

8.3.1 Overview

GaussDB(DWS) provides **gs_dump** and **gs_dumpall** to export required database objects and related information. You can use a tool to import the exported data to a target database for database migration. **gs_dump** exports a single database or its objects. **gs_dumpall** exports all databases or global objects in a cluster. For details, see [Table 8-3](#).

Table 8-3 Application scenarios

Applica tion Scenari o	Export Granularity	Export Format	Import Method
Exportin g a single databas e	Database-level export • Export full information of a database. You can use the exported information to create a same database containing the same data as the current one. • Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same database as the current one, without data. • Export data of a database.	• Plain text • Custom • Directory • .tar	• For details about how to import data files in text format, see Using a gsqr Meta-Command to Import Data.

Applica tion Scenari o	Export Granularity	Export Format	Import Method
	Schema-level export • Export full information of a schema. • Export data of a schema. • Export all object definitions of a schema, including the definitions of tables, stored procedures, and indexes. Table-level export • Export full information of a table. • Export data of a table. • Export the definition of a table.		
Exportin g all databas es in a cluster	Database-level export • Export full information of a cluster. You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one. • Export all object definitions of a cluster, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data. • Export data of a cluster. Global object export • Export tablespaces. • Export roles. • Export tablespaces and roles.	Plain text	For details about how to import data files, see Using a gsqL Meta-Command to Import Data .

gs_dump and **gs_dumpall** use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the **gs_dump** or **gs_dumpall** command to the role that has the permission. Then, **gs_dump** or **gs_dumpall** uses the specified role to export data. See [Table 8-3](#) for application scenarios and [Data Export By a User Without Required Permissions](#) for operation details.

gs_dump and **gs_dumpall** encrypt the exported data files. These files are decrypted before being imported. In this way, data disclosure is prevented, protecting database security.

When **gs_dump** or **gs_dumpall** is used to export data from a cluster, other users can still access (read and write) databases in the cluster.

gs_dump and **gs_dumpall** can export complete, consistent data. For example, if **gs_dump** is executed to export database A or **gs_dumpall** is executed to export all databases from a cluster at T1, data of database A or all databases in the cluster at that time point will be exported, and modifications on the databases after that time point will not be exported.

Obtain **gs_dump** and **gs_dumpall** by decompressing the gsql CLI client software package.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- If the number of objects (data tables, views, and indexes) in a database exceeds 500 thousand, you are advised to use **gs_guc** to set the following parameters for CNs and DN. This operation is not required if the parameter values are greater than the recommended ones.

```
gs_guc set -N all -Z coordinator -Z datanode -I all -c 'max_prepared_transactions = 1000'
gs_guc set -N all -Z coordinator -Z datanode -I all -c 'max_locks_per_transaction = 512'
```
- For data consistency and integrity, **gs_dump** and **gs_dumpall** set a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, **gs_dump** and **gs_dumpall** lock the table after it is released. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.
- During an export, **gs_dumpall** reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When you use **gsql** to execute SQL scripts, cluster administrator permissions are also required to add users and user groups, and create databases.

8.3.2 Exporting a Single Database

8.3.2.1 Exporting a Database

You can use **gs_dump** to export data and all object definitions of a database from GaussDB(DWS). You can specify the information to be exported as follows:

- Export full information of a database, including its data and all object definitions.

You can use the exported information to create a same database containing the same data as the current one.

- Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures.

You can use the exported object definitions to quickly create a same database as the current one, without data.

- Export data of a database.

Procedure

- Step 1** Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

- Step 2** Use **gs_dump** to export data of the **postgres** database.

```
gs_dump -W Bigdata@123 -U jack -f /home/dbadmin/backup/postgres_backup.tar -p 8000 postgres -F t
```

Table 8-4 Common parameters

Parameter	Description	Example Value
-U	User name for database connection.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/postgres_backup.tar
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000

Parameter	Description	Example Value
dbname	Name of the database to be exported.	postgres
-F	Format of exported files. The values of -F are as follows: • p: plain text • c: custom • d: directory • t: .tar	-F t

For details about other parameters, see [gs_dump](#).

----End

Examples

Example 1: Run **gs_dump** to export full information of the **postgres** database and compress the exported files in .sql format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/postgres_backup.sql -p 8000 postgres -Z 8 -F p
gs_dump[port='8000'][postgres][2017-07-21 15:36:13]: dump database postgres successfully
gs_dump[port='8000'][postgres][2017-07-21 15:36:13]: total time: 3793 ms
```

Example 2: Run **gs_dump** to export data of the **postgres** database, excluding object definitions. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/postgres_data_backup.dmp -p 8000 postgres -a -F c
gs_dump[port='8000'][postgres][2017-07-21 15:36:13]: dump database postgres successfully
gs_dump[port='8000'][postgres][2017-07-21 15:36:13]: total time: 3793 ms
```

Example 3: Run **gs_dump** to export object definitions of the **postgres** database. The exported files are in .sql format.

```
--Before the export, the nation table contains data.
select n_nationkey,n_name,n_regionkey from nation limit 3;
n_nationkey |      n_name      | n_regionkey
-----+-----+-----
          0 | ALGERIA          |           0
          3 | CANADA           |           1
         11 | IRAQ             |           4
(3 rows)
```

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/postgres_def_backup.sql -p 8000 postgres -s -F p
gs_dump[port='8000'][postgres][2017-07-20 15:04:14]: dump database postgres successfully
gs_dump[port='8000'][postgres][2017-07-20 15:04:14]: total time: 472 ms
```

Example 4: Run **gs_dump** to export object definitions of the **postgres** database. The exported files are in text format and are encrypted.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/postgres_def_backup.sql -p 8000 postgres --with-
encryption AES128 --with-key 1234567812345678 -s -F p
gs_dump[port='8000'][postgres][2018-11-14 11:25:18]: dump database postgres successfully
gs_dump[port='8000'][postgres][2018-11-14 11:25:18]: total time: 1161 ms
```

8.3.2.2 Exporting a Schema

You can use **gs_dump** to export data and all object definitions of a schema from GaussDB(DWS). You can export one or more specified schemas as needed. You can specify the information to be exported as follows:

- Export full information of a schema, including its data and object definitions.
- Export data of a schema, excluding its object definitions.
- Export the object definitions of a schema, including the definitions of tables, stored procedures, and indexes.

Procedure

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

Step 2 Run **gs_dump** to export the **hr** and **public** schemas.

```
gs_dump -W Bigdata@123 -U jack -f /home/dbadmin/backup/MPPDB_schema_backup -p 8000  
human_resource -n hr -F d
```

Table 8-5 Common parameters

Parameter	Description	Example Value
-U	User name for database connection.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/MPPDB_schema_backup
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000

Parameter	Description	Example Value
dbname	Name of the database to be exported.	human_resource
-n	Names of schemas to be exported. Data of the specified schemas will also be exported. - Single schema: Enter -n <i>schemaname</i>. - Multiple schemas: Enter -n <i>schemaname</i> for each schema.	- Single schemas: -n hr - Multiple schemas: -n hr -n public
-F	Format of exported files. The values of -F are as follows: p: plain text c: custom d: directory t: .tar	-F d

For details about other parameters, see [gs_dump](#).

----End

Examples

Example 1: Run **gs_dump** to export full information of the **hr** schema. The exported files are compressed and stored in text format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_backup.sql -p 8000 human_resource
-n hr -Z 6 -F p
gs_dump[port='8000'][human_resource][2017-07-21 16:05:55]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:05:55]: total time: 2425 ms
```

Example 2: Run **gs_dump** to export data of the **hr** schema. The exported files are in .tar format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_data_backup.tar -p 8000
human_resource -n hr -a -F t
gs_dump[port='8000'][human_resource][2018-11-14 15:07:16]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-11-14 15:07:16]: total time: 1865 ms
```

Example 3: Run **gs_dump** to export the object definitions of the **hr** schema. The exported files are stored in a directory.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_def_backup -p 8000
human_resource -n hr -s -F d
gs_dump[port='8000'][human_resource][2018-11-14 15:11:34]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-11-14 15:11:34]: total time: 1652 ms
```

Example 4: Run **gs_dump** to export the **human_resource** database excluding the **hr** schema. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_backup.dmp -p 8000
human_resource -N hr -F c
gs_dump[port='8000'][human_resource][2017-07-21 16:06:31]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:06:31]: total time: 2522 ms
```

Example 5: Run **gs_dump** to export the object definitions of the **hr** and **public** schemas, encrypt the exported files, and store them in .tar format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_backup1.tar -p 8000
human_resource -n hr -n public -s --with-encryption AES128 --with-key 1234567812345678 -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:07:16]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:07:16]: total time: 2132 ms
```

Example 6: Run **gs_dump** to export the **human_resource** database excluding the **hr** and **public** schemas. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_schema_backup2.dmp -p 8000
human_resource -N hr -N public -F c
gs_dump[port='8000'][human_resource][2017-07-21 16:07:55]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:07:55]: total time: 2296 ms
```

Example 7: Run **gs_dump** to export all tables (views, sequences, and foreign tables are also included) in the **public** schema and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_backup3.dmp -p 8000 human_resource -t
public.* -t hr.staffs -F c
gs_dump[port='8000'][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

8.3.2.3 Exporting a Table

You can use **gs_dump** to export data and definition of a table-level object from GaussDB(DWS). Views, sequences, and foreign tables are special tables. You can export one or more specified tables as needed. You can specify the information to be exported as follows:

- Export full information of a table, including its data and definition.
- Export data of a table.
- Export the definition of a table.

Procedure

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

Step 2 Run **gs_dump** to export the **hr.staffs** and **hr employments** tables.

```
gs_dump -W Bigdata@123 -U jack -f /home/dbadmin/backup/MPPDB_table_backup -p 8000
human_resource -t hr.staffs -F d
```

Table 8-6 Common parameters

Parameter	Description	Example Value
-U	User name for database connection.	-U jack

Parameter	Description	Example Value
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/MPPDB_table_backup
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000
dbname	Name of the database to be exported.	human_resource
-t	Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters. When you use wildcard characters, quote wildcard patterns with single quotation marks (") to prevent the shell from expanding the wildcard characters. - Single table: Enter -t <i>schema.table</i>. - Multiple tables: Enter -t <i>schema.table</i> for each table.	- Single table: -t <i>hr.staffs</i> - Multiple tables: -t <i>hr.staffs</i> -t <i>hr employments</i>
-F	Format of exported files. The values of -F are as follows: p: plain text c: custom d: directory t: .tar	-F d

For details about other parameters, see [gs_dump](#).

----End

Examples

Example 1: Run **gs_dump** to export full information of the **hr.staffs** table. The exported files are in text format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup.sql -p 8000 human_resource -t hr.staffs -Z 6 -F p
gs_dump[port='8000'][human_resource][2017-07-21 17:05:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:05:10]: total time: 3116 ms
```

Example 2: Run **gs_dump** to export data of the **hr.staffs** table. The exported files are in .tar format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_data_backup.tar -p 8000 human_resource -t hr.staffs -a -F t
gs_dump[port='8000'][human_resource][2017-07-21 17:04:26]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:04:26]: total time: 2570 ms
```

Example 3: Run **gs_dump** to export the definition of the **hr.staffs** table. The exported files are stored in a directory.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_def_backup -p 8000 human_resource -t hr.staffs -s -F d
gs_dump[port='8000'][human_resource][2017-07-21 17:03:09]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:03:09]: total time: 2297 ms
```

Example 4: Run **gs_dump** to export the **human_resource** database excluding the **hr.staffs** table. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup4.dmp -p 8000 human_resource -T hr.staffs -F c
gs_dump[port='8000'][human_resource][2017-07-21 17:14:11]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:14:11]: total time: 2450 ms
```

Example 5: Run **gs_dump** to export the **hr.staffs** and **hr employments** tables. The exported files are in text format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup1.sql -p 8000 human_resource -t hr.staffs -t hr.employments -F p
gs_dump[port='8000'][human_resource][2017-07-21 17:19:42]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:19:42]: total time: 2414 ms
```

Example 6: Run **gs_dump** to export the **human_resource** database excluding the **hr.staffs** and **hr employments** tables. The exported files are in text format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup2.sql -p 8000 human_resource -T hr.staffs -T hr.employments -F p
gs_dump[port='8000'][human_resource][2017-07-21 17:21:02]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 17:21:02]: total time: 3165 ms
```

Example 7: Run **gs_dump** to export data and definition of the **hr.staffs** table, and the definition of the **hr employments** table. The exported files are in .tar format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup3.tar -p 8000 human_resource -t hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
gs_dump[port='8000'][human_resource][2018-11-14 11:32:02]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-11-14 11:32:02]: total time: 1645 ms
```

Example 8: Run **gs_dump** to export data and definition of the **hr.staffs** table, encrypt the exported files, and store them in text format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup4.sql -p 8000 human_resource -t hr.staffs --with-encryption AES128 --with-key 1212121212121212 -F p
gs_dump[port='8000'][human_resource][2018-11-14 11:35:30]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-11-14 11:35:30]: total time: 6708 ms
```

Example 9: Run **gs_dump** to export all tables (views, sequences, and foreign tables are also included) in the **public** schema and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -f /home/dbadmin/backup/MPPDB_table_backup5.dmp -p 8000 human_resource
-t public.* -t hr.staffs -F c
gs_dump[port='8000'][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

Example 10: Run **gs_dump** to export the definition of the view referencing to the **test1** table in the **t1** schema. The exported files are in a custom format.

```
gs_dump -W Bigdata@123 -U jack -f /home/dbadmin/backup/MPPDB_view_backup6 -p 8000
human_resource -t t1.test1 --include-depend-objs --exclude-self -F d
gs_dump[port='8000'][jack][2018-11-14 17:21:18]: dump database human_resource successfully
gs_dump[port='8000'][jack][2018-11-14 17:21:23]: total time: 4239 ms
```

8.3.3 Exporting All Databases

8.3.3.1 Exporting All Databases

You can use **gs_dumpall** to export full information of all databases in a cluster from GaussDB(DWS), including information about each database and global objects in the cluster. You can specify the information to be exported as follows:

- Export full information of all databases, including information about each database and global objects (such as roles and tablespaces) in the cluster.
You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one.
- Export data of all databases, excluding all object definitions and global objects.
- Export all object definitions of all databases, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures.

You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data.

Procedure

- Step 1** Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

- Step 2** Run **gs_dumpall** to export full information of all databases.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000
```

Table 8-7 Common parameters

Parameter	Description	Example Value
-U	User name for database connection. The user must be a cluster administrator.	-U omm
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/MPPDB_backup.sql
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000

For details about other parameters, see [gs_dumpall](#).

----End

Examples

Example 1: Run **gs_dumpall** as a cluster administrator **omm** to export full information of all databases in a cluster. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the backup is successful. In this example, only relative output information is included.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000
gs_dumpall[port='8000'][2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port='8000'][2017-07-21 15:57:31]: total time: 9627 ms
```

Example 2: Run **gs_dumpall** as a cluster administrator **omm** to export object definitions of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the backup is successful. In this example, only relative output information is included.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -s
gs_dumpall[port='8000'][2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port='8000'][2018-11-14 11:28:14]: total time: 4147 ms
```

Example 3: Run **gs_dumpall** to export data of all databases in a cluster, encrypt the exported files, and store them in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the backup is successful. In this example, only relative output information is included.

```
gs_dumpall -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -a --with-encryption AES128 --with-key
1234567812345678
gs_dumpall[port='8000'][2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port='8000'][2018-11-14 11:23:26]: total time: 4147 ms
```

8.3.3.2 Exporting Global Objects

You can use **gs_dumpall** to export global objects, including database users, user groups, tablespaces, and attributes (for example, global access permissions), from GaussDB(DWS).

Procedure

- Step 1** Connect to the default database **postgres** as a database administrator through the database client tool provided by DWS.

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

- Step 2** Run **gs_dumpall** to export global tablespaces.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -t
```

Table 8-8 Common parameters

Parameter	Description	Example Value
-U	User name for database connection. The user must be a cluster administrator.	-U omm
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123

Parameter	Description	Example Value
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/ <i>MPPDB_tablespace.sql</i>
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000
-t	Dumps only tablespaces. You can also use --tablespaces-only alternatively.	-

For details about other parameters, see [gs_dumpall](#).

----End

Examples

Example 1: Run **gs_dumpall** as a cluster administrator **omm** to export global tablespaces and users in a cluster. The exported files are in text format.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_globals.sql -p 8000 -g
gs_dumpall[port='8000'][2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port='8000'][2018-11-14 19:06:24]: total time: 1150 ms
```

Example 2: Run **gs_dumpall** to export global tablespaces in a cluster, encrypt the exported files, and store them in text format as a cluster administrator **omm**.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -t --with-encryption AES128 --with-key 1212121212121212
gs_dumpall[port='8000'][2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port='8000'][2018-11-14 19:00:58]: total time: 186 ms
```

Example 3: Run **gs_dumpall** as a cluster administrator **omm** to export global users in a cluster. The exported files are in text format.

```
gs_dumpall -W Bigdata@123 -U omm -f /home/dbadmin/backup/MPPDB_user.sql -p 8000 -r
gs_dumpall[port='8000'][2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port='8000'][2018-11-14 19:03:18]: total time: 162 ms
```

8.3.4 Data Export By a User Without Required Permissions

gs_dump and **gs_dumpall** use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the **gs_dump** or **gs_dumpall** command to the role that has the permission. Then, **gs_dump** or **gs_dumpall** uses the specified role to export data.

Procedure

- Step 1** Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

Step 2 Use **gs_dump** to export data of the **human_resource** database.

User **jack** does not have the permission to export data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the **gs_dump** command. The exported files are in .tar format.

```
gs_dump -U jack -W Bigdata@234 -f /home/dbadmin/backup/MPPDB_backup.tar -p 8000 human_resource  
--role role1 --rolepassword abc@1234 -F t
```

Table 8-9 Common parameters

Parameter	Description	Example Value
-U	User name for database connection.	-U jack
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Bigdata@123
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/MPPDB_backup.tar
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000
dbname	Name of the database to be exported.	human_resource

Parameter	Description	Example Value
--role	Role name for the export operation. After this parameter is set, the SET ROLE command will be issued after gs_dump or gs_dumpall connects to the database. It is useful when the user specified by -U does not have the permissions required by gs_dump or gs_dumpall . This parameter allows you to switch to a role with the required permissions.	-r role1
--rolepassword	Role password.	--rolepassword abc@1234
-F	Format of exported files. The values of -F are as follows: • p: plain text • c: custom • d: directory • t: .tar	-F t

For details about other parameters, see [gs_dump](#) or [gs_dumpall](#).

-----End

Examples

Example 1: User **jack** does not have the permission to export data of the **human_resource** database using **gs_dump** and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the **gs_dump** command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";
CREATE USER

gs_dump -U jack -W 1234@abc -f /home/dbadmin/backup/MPPDB_backup11.tar -p 8000 human_resource --
role role1 --rolepassword abc@1234 -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 4239 ms
```

Example 2: User **jack** does not have the permission to export the **public** schema using **gs_dump** and the role **role1** has this permission. To export the **public** schema, you can set **--role** to **role1** in the **gs_dump** command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";
CREATE USER

gs_dump -U jack -W 1234@abc -f /home/dbadmin/backup/MPPDB_backup12.tar -p 8000 human_resource -
n public --role role1 --rolepassword abc@1234 -F t
```

```
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully  
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 3278 ms
```

Example 3: User **jack** does not have the permission to export all databases in a cluster using **gs_dumpall** and the role **role1** (cluster administrator) has this permission. To all the databases, you can set **--role** to **role1** in the **gs_dumpall** command. The exported files are in text format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";  
CREATE USER
```

```
gs_dumpall -U jack -W 1234@abc -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 --role role1 --  
rolepassword abc@1234  
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: dumpall operation successful  
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: total time: 6437 ms
```

9 Querying External Data

9.1 Querying Data on OBS

9.1.1 OBS Query Overview

GaussDB(DWS) provides the SQL on OBS feature that allows users to directly use **SELECT** to query TEXT, CSV, ORC, or CARBONDATA data stored on OBS through a foreign table, so that they do not need to import OBS data to a data warehouse cluster and query it. In this way, query time and cluster resource consumption are greatly reduced. This chapter describes how to query for ORC data stored in OBS as an example.

NOTE

When querying OBS data through a foreign table, transaction tables in the CARBONDATA format generated by DLI are not supported. If the table is not a transaction table, you can query it through an OBS foreign table.

9.1.2 Preparing Data on OBS

Scenarios

Before you use the SQL on OBS feature to query OBS data:

1. You have stored the ORC data on OBS.
For example, the ORC table has been created when you use the Hive or Spark component, and the ORC data has been stored on OBS.
Assume that there are two ORC data files, named **product_info.0** and **product_info.1**, whose original data is stored in the **demo.db/product_info_orc/** directory of the **mybucket** OBS bucket. You can view their original data in [Original Data](#).
2. If your data files are already on OBS, perform steps in [Obtaining the OBS Path of Original Data and Setting Read Permission](#).

Original Data

Assume that you have stored the two ORC data files on OBS and their original data is as follows:

- Data file **product_info.0**

The sample data is provided as follows:

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,
328,2017-09-04,715,good!
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,
584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's
really super nice.
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,
1223,2017-09-06,1200,The seller's packaging is exquisite.
```

- Data file **product_info.1**

The sample data is provided as follows:

```
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,
997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,
841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's
really amazing to buy.
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,
114,2017-09-14,22,Open the package and the clothes have no odor.
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,
2004,2017-09-15,826,Very favorite clothes.
```

Obtaining the OBS Path of Original Data and Setting Read Permission

Step 1 Log in to the OBS management console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console.

Alternatively, you can access <https://storage.huaweicloud.com> to log in to the OBS management console.

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. You need to specify the OBS paths of source data files when creating a foreign table.

For example, the OBS paths are as follows:

```
https://obs.cn-north-1.myhuaweicloud.com/mybucket/demo.db/product_info_orc/product_info.0
https://obs.cn-north-1.myhuaweicloud.com/mybucket/demo.db/product_info_orc/product_info.1
```

Step 3 Grant the OBS bucket read permission for the user.

The user who executes the SQL on OBS function needs to obtain the read permission on the OBS bucket where the source data file is located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

----End

9.1.3 Creating a Foreign Server

This section describes how to create a foreign server that is used to define the information about the OBS server and is invoked by foreign tables. For details about the syntax for creating foreign servers, see CREATE SERVER.

(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions

Common users do not have permissions to create foreign servers and tables. If you want to use a common user to create foreign servers and tables in a customized database, perform the following steps to create a user and a database, and grant the user foreign table permissions.

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

- Step 1** Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -r
```

Enter your password as prompted.

- Step 2** Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'Bigdata@123';
```

Switch to the created user.

```
SET ROLE dbuser PASSWORD 'Bigdata@123';
```

Run the following command to create the database demo:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

datname	datdba	encoding	datcollate	datctype	datistemplate	dataallowconn	datconnlimit	datacl
template1	10	0	C	C	t	t	-1	14146 1351
1663 ORA			{=c/omm,omm=CTc/omm}					
template0	10	0	C	C	t	f	-1	14146 1350
1663 ORA			{=c/omm,omm=CTc/omm}					
postgres	10	0	C	C	f	t	-1	14146 1352
1663 ORA			{=Tc/omm,omm=CTc/omm,chaojun=C/omm,huobinru=C/omm}					
mydatabase	17000	0	C	C	f	t	-1	14146 1351
1663 ORA								

(4 rows)

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Connect to the new database as a database administrator through the database client tool provided by GaussDB(DWS).

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password of the system administrator as prompted.

NOTE

Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON FOREIGN DATA WRAPPER dfs_fdw TO dbuser;
```

where *fdw_name* can be **hdfs_fdw** or **dfs_fdw**, and **dbuser** is the name of the user who creates SERVER.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
       r.rolcreatorrole, r.rolcreatedb, r.rolcanlogin,
       r.rolconndeflimit, r.rolvalidbegin, r.rolvaliduntil,
       ARRAY(SELECT b.rolname
             FROM pg_catalog.pg_auth_members m
             JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
             WHERE m.member = r.oid) as memberof
       , r.rolreplication
       , r.rolauditadmin
       , r.rolsystemadmin
       , r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the UseFT permission.

rolname	rolsuper	rolinherit	rolcreatorrole	rolcreatedb	rolcanlogin	rolconndeflimit	rolvalidbegin	rolvaliduntil	memberof	rolreplication	rolauditadmin	rolsystemadmin	roluseft
dbuser	f	t	f	t	t	-1			{}	f			
lily	f	t	f	f	t	-1			{}	f			
omm	t	t	t	t	t	-1			{}				
t	t	t	t	t									

----End

Creating a Foreign Server

Step 1 Use the user who is about to create a foreign server to connect to the corresponding database.

In this example, use common user **dbuser** created in [\(Optional\) Creating a User and a Database and Granting the User Foreign Table Permissions](#) to connect to **mydatabase** created by the user. You need to connect to the database through the database client tool provided by GaussDB(DWS).

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

```
\c mydatabase dbuser;
```


Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:

```
gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
```


Enter the password as prompted.

Step 2 Create a foreign server.

For details about the syntax for creating foreign servers, see CREATE SERVER.

For example, run the following command to create a foreign server named **obs_server**.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER dfs_fdw  
OPTIONS (  
  address 'obs.cn-north-1.myhuaweicloud.com' ,  
  ACCESS_KEY 'access_key_value_to_be_replaced',  
  SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',  
  type 'obs'  
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*
You can customize a name.
In this example, the name is set to **obs_server**.
- **FOREIGN DATA WRAPPER**
fdw_name can be **hdfs_fdw** or **dfs_fdw**, which already exists in the database.
- **OPTIONS parameters**
 - **address**
This parameter specifies the IP address or domain name of the OBS server.
Obtain the address as follows:
 - Obtain the OBS path by performing [2](#) in [Preparing Data on OBS](#).
 - An OBS path is displayed in either of the following modes, which include the IP address or domain name of OBS:
 - https://IP address or domain name of the storage server/Bucket name/Directory level/Object name

- ### Step 3 View the foreign server.

The server is successfully created if the returned result is as follows:

----End

After steps in [Creating a Foreign Server](#) are performed, create an OBS foreign table in the GaussDB(DWS) database to access the data stored on OBS. An OBS foreign table is read-only. It can only be queried using **SELECT**.

The syntax for creating a foreign table is as follows. For details, see the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

For example, when creating a foreign table named *product_info_ext_obs*, set parameters in the syntax as follows:

- **table_name**
Specifies the name of the foreign table to be created.
- **Table column definitions**
 - **column_name**: specifies the name of a column in the foreign table.
 - **type_name**: specifies the data type of the column.

Multiple columns are separate by commas (,).

The number of fields and field types in the foreign table must be the same as those in the data stored on OBS.
- **SERVER dfs_server**
This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign server connects to OBS to read data by setting its foreign server.
Enter the name of the foreign server created by following steps in [Creating a Foreign Server](#).
- **OPTIONS parameters**
These are parameters associated with the foreign table. The key parameters are as follows:
 - **format**: indicates the file format on OBS. The ORC and CARBONDATA formats are supported.
 - **foldername**: This parameter is mandatory. It specifies the OBS path of the source data file. You only need to enter */Bucket name/Directory level*, which does not contain the IP address or domain name of OBS.
You can perform 2 in [Preparing Data on OBS](#) to obtain the complete OBS path of the source data file. The path may be in either of the following formats, which contain the bucket name and the directory level.
 - *https://IP address or domain name of the storage server/Bucket name/Directory level/Object name*
 - *https://Bucket name.Domain name/Directory level/Object name*
 - **totalrows**: This parameter is optional. It does not indicate the total rows of the imported data. Because OBS may store many files, it is slow to analyze data. This parameter allows you to set an estimated value so that the optimizer can estimate the table size according to the value. Generally, query efficiency is relatively high when the estimated value is almost the same as the actual value.
 - **encoding**: encoding of data source files in foreign tables. The default value is **utf8**. This parameter is mandatory for OBS foreign tables.
- **DISTRIBUTE BY:**
This clause is mandatory. Currently, OBS foreign tables support only the **ROUNDROBIN** distribution mode.
It indicates that when a foreign table reads data from the data source, each node in the GaussDB(DWS) cluster randomly reads some data and integrates the random data to a complete data set.
- **Other parameters in the syntax**
Other parameters are optional. You can set them as required. In this example, you do not need to set these parameters. For details, see **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;  
  
-- Create an OBS foreign table that does not contain partition columns. The foreign server associated with  
the table is obs_server, the file format on OBS corresponding to the table is ORC, and the data storage  
path on OBS is /mybucket/data/.  
  
CREATE FOREIGN TABLE product_info_ext_obs  
(  
    product_price          integer      not null,  
    product_id             char(30)     not null,  
    product_time           date         ,  
    product_level          char(10)     ,  
    product_name           varchar(200) ,  
    product_type1          varchar(20)  ,  
    product_type2          char(10)     ,  
    product_monthly_sales_cnt integer    ,  
    product_comment_time   date         ,  
    product_comment_num    integer      ,  
    product_comment_content varchar(200)  
) SERVER obs_server  
OPTIONS (  
    format 'orc',  
    foldername '/mybucket/demo.db/product_info_orc/',  
    encoding 'utf8',  
    totalrows '10'  
)  
DISTRIBUTE BY ROUNDROBIN;
```

9.1.5 Querying Data on OBS Through Foreign Tables

Viewing Data on OBS by Directly Querying the Foreign Table

If the data amount is small, you can directly run **SELECT** to query the foreign table and view the data on OBS.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM product_info_ext_obs;
```

If the query result is the same as the data in [Original Data](#), the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

Step 1 Create a table in GaussDB(DWS) to store imported data.

The target table structure must be the same as the structure of the foreign table created in [Creating a Foreign Table](#). That is, the number of fields and field types must be the same.

For example, create a table named *product_info*. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;  
  
CREATE TABLE product_info
```

```
(
  product_price      integer      not null,
  product_id         char(30)     not null,
  product_time       date        ,
  product_level      char(10)    ,
  product_name       varchar(200) ,
  product_type1      varchar(20) ,
  product_type2      char(10)    ,
  product_monthly_sales_cnt integer ,
  product_comment_time date      ,
  product_comment_num integer    ,
  product_comment_content varchar(200)
)
with (
  orientation = column,
  compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Run the **INSERT INTO.. SELECT ..** command to import data from the foreign table to the target table.

Example:

```
INSERT INTO product_info SELECT * FROM product_info_ext_obs;
```

If information similar to the following is displayed, the data has been imported.
INSERT 0 10

Step 3 Run the following **SELECT** command to view data imported from OBS to GaussDB(DWS):

```
SELECT * FROM product_info;
```

If the query result displays data specified in **Original Data**, the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

```
----End
```

9.1.6 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation. The procedure is as follows:

1. **Deleting the Foreign Table and Target Table**
2. **Deleting the Created Foreign Server**
3. **Deleting the Database and the User to Which the Database Belongs**

If you have performed steps in **(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions**, delete the database and the user to which the database belongs.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If **Querying Data After Importing It** is performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

If the following information is displayed, the table has been deleted.

```
DROP TABLE
```

Step 2 Run the following statement to delete the foreign table:

```
DROP FOREIGN TABLE product_info_ext_obs;
```

If the following information is displayed, the table has been deleted.

```
DROP FOREIGN TABLE
```

----End

Deleting the Created Foreign Server

Step 1 Use the user who created the foreign server to connect to the database where the foreign server is located.

In this example, common user **dbuser** is used to create the foreign server in **mydatabase**. You need to connect to the database through the database client tool provided by GaussDB(DWS). You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the **gsql** client, run the following command to switch the database and user:

```
\c mydatabase dbuser;
```

Enter the password as prompted.

- If you have logged in to the **gsql** client, you can run the **\q** command to exit **gsql**, and run the following command to reconnect to it:

```
gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
```

Enter the password as prompted.

Step 2 Delete the created foreign server.

Run the following command to delete the server. For details about the syntax, see **DROP SERVER**.

```
DROP SERVER obs_server;
```

The database is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

The server is successfully deleted if the returned result is as follows:

```
srvname | srvowner | srfdw | srvtype | srvversion | srval | srvoptions  
-----+-----+-----+-----+-----+-----+-----  
(0 rows)
```

----End

Deleting the Database and the User to Which the Database Belongs

If you have performed steps in [\(Optional\) Creating a User and a Database and Granting the User Foreign Table Permissions](#), perform the following steps to delete the database and the user to which the database belongs.

Step 1 Delete the customized database.

Connect to the default database **postgres** through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

Switch to the default database **postgres**.

```
\c postgres
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:

```
DROP DATABASE
```

Step 2 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:

```
REVOKE ALL ON FOREIGN DATA WRAPPER dfs_fdw FROM dbuser;  
REVOKE
```

The name of **FOREIGN DATA WRAPPER** must be **dfs_fdw**. **dbuser** is the user name for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;  
DROP USER
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

9.2 Querying Data on MRS

On GaussDB(DWS), you can remotely read MRS data and import data from MRS to GaussDB(DWS) for query. For details, see [Importing Data From MRS to a Data Warehouse Cluster](#).

10 Using Express to Query Data on OBS

10.1 Express Functions

This chapter describes how to use Express clusters to provide services to data warehouse clusters, create and query tables, and check whether a data warehouse cluster is accelerated.

The Express cluster provides computing acceleration for data warehouse clusters. It is enabled by default. This function is charged based on the bytes returned from the Express cluster to a data warehouse cluster. For details, see [GaussDB\(DWS\) Pricing Details](#).

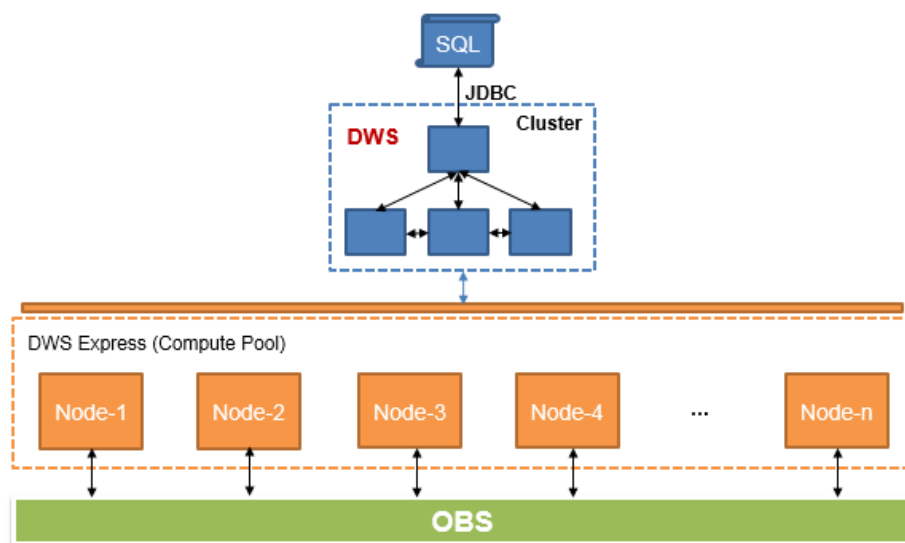
Overview

Express clusters are independent of data warehouse clusters but are located at the same level as the data warehouse clusters. An Express cluster provides computing acceleration services for one or more data warehouse clusters. By default, Express clusters provide services for all data warehouse clusters of the matched version.

During the actual use of data warehouse clusters, the aged service data is put into OBS for joint queries with internal GaussDB(DWS) tables. Foreign table querying of GaussDB(DWS) uses the full-table scanning mode. The processing speed depends on the cluster performance. If a data warehouse cluster has a small number of nodes and low specifications, the processing speed is slow.

The Express cluster is a shared cluster that speeds up the query on OBS foreign tables. When you perform OBS foreign table querying in a data warehouse cluster, the data scanning operator, filtering operator, and aggregation operator are pushed to the Express cluster for calculation. The Express cluster has a larger scale and higher performance than the user cluster, so a better acceleration effect can be achieved.

You do not need to modify existing services when using the Express cluster. When you perform OBS foreign table querying, the system automatically determines whether to execute Express query acceleration based on the resource usage. If the acceleration is required, the execution plan of operators such as data scanning, filtering, and aggregation is pushed to the Express cluster. After the calculation is complete, the Express cluster returns the data to the data warehouse cluster, which performs subsequent queries based on the execution plan.

Figure 10-1 Express architecture

10.2 Using Express Clusters

10.2.1 Example

The following example shows how to query the Express sample data stored in an OBS bucket using foreign tables and check whether the query is accelerated. In this example, the Express sample data set has been generated on GaussDB(DWS) and uploaded to the **tpch-obs-express** folder of an OBS bucket. All users have been granted the read-only permission to access the OBS bucket. They can easily import the data set using their accounts.

10.2.2 Creating Foreign Servers and OBS Foreign Tables

OBS foreign tables need to be created on foreign servers. To create a foreign server, connect to the IP address or domain name of OBS by using your AK and SK. After creating a foreign server, create a foreign table and associate it with the Express sample data uploaded to OBS.

After connecting to the cluster using the SQL client tool, you can run SQL statements in the SQL client tool to create foreign servers, part tables, and lineitem tables.

Step 1 Use the SQL client tool to connect to the GaussDB(DWS) cluster.

For details about how to connect to the cluster, see section [Using the Data Studio GUI Client to Connect to a Cluster](#).

Step 2 Create a foreign server.

The statement for creating a foreign server is as follows. Replace *<OBS_Endpoint>*, *<Access_Key_Id>*, and *<Secret_Access_Key>* with the actual values according to the parameter descriptions below the statement, and then run the statement to create a foreign server.


```
CREATE SERVER express_server
FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
    address '<OBS_Endpoint>',
    type 'obs',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>'
);
```

Parameter description:

- **<OBS_Endpoint>**: Enter the endpoint address of OBS.
For more information about the endpoint, visit [Regions and Endpoints](#).
- **<Access_Key_Id>** and **<Secret_Access_Key>**: Indicate the access key ID (AK) and secret access key (SK).
For more information about the AK and SK, see [Creating Access Keys \(AK and SK\)](#).

Step 3 Create an OBS foreign table.

The statement for creating an OBS foreign table is as follows, in which **<obs_bucket_name>** indicates the name of the OBS bucket for storing sample data. The bucket name varies with the region. Based on the region where the cluster resides, replace parameter **<obs_bucket_name>** with the OBS bucket name in the corresponding region, and run the statement to create an OBS foreign table. The OBS name in **CN North-Beijing1** is **dws-demo**, and the OBS name in other regions is **dws-demo-<Region>**, in which **<Region>** indicates the region name. For information about the regions, see [Regions and Endpoints](#). For example, if the region is **cn-south-1**, **<obs_bucket_name>** is **dws-demo-cn-south-1**.

```
CREATE schema tpchobs_express;
set current_schema = 'tpchobs_express';
CREATE FOREIGN TABLE PART
(
    P_PARTKEY    BIGINT NOT NULL,
    P_NAME       VARCHAR(55) NOT NULL,
    P_MFGR       CHAR(25) NOT NULL,
    P_BRAND      CHAR(10) NOT NULL,
    P_TYPE       VARCHAR(25) NOT NULL,
    P_SIZE       BIGINT NOT NULL,
    P_CONTAINER  CHAR(10) NOT NULL,
    P_RETAILPRICE DECIMAL(15,2) NOT NULL,
    P_COMMENT    VARCHAR(23) NOT NULL
)
SERVER express_server
OPTIONS(
    encoding 'utf-8',
    format 'text',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    foldername '/<obs_bucket_name>/tpch-obs-express/part/'
)
distribute by roundrobin;
```

----End

10.2.3 Querying Data

Query the number of table records. Run the count statement to query the number of table records.

```
set current_schema='tpchobs_express';
select count(*) from part;
```

The result is as follows:

```
count
-----
20000
(1 row)
```

10.2.4 Checking Whether a Query Is Accelerated

To check whether the current query is accelerated, run the explain command to view the execution plan.

```
set current_schema='tpchobs_express';
explain select count(*) from part;
```

The command output is as follows. If the command output contains **PLAN ROUTER**, the query is accelerated.

```
id | operation | E-rows | E-width | E-costs
+-----+-----+-----+-----+-----+
1 | -> Row Adapter | 1 | 8 | 182.30
2 | -> Vector Aggregate | 1 | 8 | 182.30
3 | -> Vector Streaming (type: GATHER) | 6 | 8 | 182.30
4 | -> Vector Streaming (type: PLAN ROUTER) | 6 | 8 | 182.08
5 | -> Vector Streaming (type: SCAN GATHER) | 6 | 8 | 182.08
6 | -> Vector Aggregate | 6 | 8 | 182.08
7 | -> Vector Foreign Scan on part
  3413 | 0 | 180.65
(7 rows)
Predicate Information (identified by plan id)
-----
 7 --Vector Foreign Scan on part
    Server Type: obs
(2 rows)
```

- Note:

Query acceleration means that the current query is pushed down to the Express cluster for execution. To view the data volumes pushed down to the Express cluster, query system table **gs_obsscaninfo**. The query example is as follows:

```
select * from gs_obsscaninfo order by time_stamp desc;
```

The query result is shown in the following figure. The **data_size** column in the table stores all the scanning volumes (in bytes) pushed down to the Express cluster in the current query.

Figure 10-2 Query result

query_id	user_id	table_name	file_type	time_stamp	actual_time	file_scanned	data_size	billing_info
72628543992254385	dbadmin	stocks_day30	csv	2019-05-22 07:35:48.145812+00	.165651	80	0	unbilled
72628543992117237	dbadmin	stocks_day30	csv	2019-05-22 07:35:45.730126+00	.169957	80	0	unbilled
72628543992117189	dbadmin	stocks_day30	csv	2019-05-22 07:35:03.331944+00	.171408	80	0	unbilled
72628543992254325	dbadmin	stocks_day30	csv	2019-05-22 07:34:46.544304+00	.16799	80	0	unbilled
72628543992117170	dbadmin	stocks_day30	csv	2019-05-22 07:34:44.171827+00	.141521	80	0	unbilled
72628543992117157	dbadmin	stocks_day30	csv	2019-05-22 07:34:20.448411+00	.185163	80	0	unbilled
72628543992254302	dbadmin	stocks_day30	csv	2019-05-22 07:34:12.883403+00	.142191	80	0	unbilled
72628543992117142	dbadmin	stocks_day30	csv	2019-05-22 07:34:05.454077+00	.220111	80	0	unbilled
72628543992254287	dbadmin	stocks_day30	csv	2019-05-22 07:33:56.78273+00	.167754	80	0	unbilled
72628543992117129	dbadmin	stocks_day30	csv	2019-05-22 07:33:45.992336+00	.122993	80	0	unbilled
72628543992117112	dbadmin	stocks_day30	csv	2019-05-22 07:33:30.984229+00	1.646737	80	525641948	unbilled
72628543992254167	dbadmin	stocks_day30	csv	2019-05-22 07:31:46.055983+00	.164234	80	0	unbilled
72628543992117036	dbadmin	stocks_day30	csv	2019-05-22 07:31:43.542376+00	.172294	80	0	unbilled
72628543992254152	dbadmin	stocks_day30	csv	2019-05-22 07:31:30.998236+00	.136751	80	0	unbilled
72628543992117013	dbadmin	stocks_day30	csv	2019-05-22 07:31:07.421039+00	1.335934	80	525640996	unbilled
72628543992254129	dbadmin	stocks_day30	csv	2019-05-22 07:30:55.607438+00	1.383235	80	525641948	unbilled
72628543992116998	dbadmin	stocks_day30	csv	2019-05-22 07:30:52.450513+00	.238344	80	0	unbilled
72628543992254114	dbadmin	stocks_day30	csv	2019-05-22 07:30:40.596262+00	.147539	80	0	unbilled
72628543992254097	dbadmin	stocks_day30	csv	2019-05-22 07:30:20.930224+00	1.547364	80	525640996	unbilled
72628543992116948	dbadmin	stocks_day30	csv	2019-05-22 07:30:06.633065+00	1.683422	80	525640996	unbilled
72628543992254003	dbadmin	stocks_day30	csv	2019-05-22 07:28:26.615992+00	1.812155	80	525641948	unbilled
72628543992116870	dbadmin	stocks_day30	csv	2019-05-22 07:28:19.185716+00	2.829206	80	1051202854	unbilled
72628543992253857	dbadmin	stocks_day30	csv	2019-05-22 07:28:12.258306+00	1.844833	80	1051202854	unbilled
72628543992116851	dbadmin	stocks_day30	csv	2019-05-22 07:27:54.795065+00	1.547422	80	1051202854	unbilled

11 PostGIS Extension

11.1 PostGIS

GaussDB(DWS) provides PostGIS Extension (PostGIS-2.4.2). PostGIS Extension is a spatial database extender for PostgreSQL. It provides the following spatial information services: spatial objects, spatial indexes, spatial functions, and spatial operators. PostGIS Extension complies with the OpenGIS specifications.

In GaussDB(DWS), PostGIS Extension depends on the listed third-party open-source software.

- Geos 3.6.2
- Proj 4.9.2
- Json 0.12.1
- Libxml2 2.7.1
- Gdal 1.11.0

11.2 Using PostGIS

Creating PostGIS Extension

Run the **CREATE EXTENSION** command to create PostGIS Extension.

```
CREATE EXTENSION postgis;
```

After PostGIS Extension is created, you can run the following command to enable the grid function in PostGIS if needed:

```
CREATE EXTENSION postgis_raster;
```

Using PostGIS Extension

Use the following function to invoke a PostGIS Extension:

```
SELECT GisFunction (Param1, Param2,.....);
```

GisFunction is the function, and **Param1** and **Param2** are function parameters. The following SQL statements are a simple illustration for PostGIS use. For details about related functions, see [PostGIS 2.4.2 Manual](#).

Example 1: Create a geometry table.

```
CREATE TABLE cities ( id integer, city_name varchar(50) );  
SELECT AddGeometryColumn('cities', 'position', 4326, 'POINT', 2);
```

Example 2: Insert geometry data.

```
INSERT INTO cities (id, position, city_name) VALUES (1,ST_GeomFromText('POINT(-9.5 23)',4326),'CityA');  
INSERT INTO cities (id, position, city_name) VALUES (2,ST_GeomFromText('POINT(-10.6 40.3)',4326),'CityB');  
INSERT INTO cities (id, position, city_name) VALUES (3,ST_GeomFromText('POINT(20.8 30.3)',4326), 'CityC');
```

Example 3: Calculate the distance between any two cities among three cities.

```
SELECT p1.city_name,p2.city_name,ST_Distance(p1.position,p2.position) FROM cities AS p1, cities AS p2  
WHERE p1.id > p2.id;
```

Deleting PostGIS Extension

Run the following command to delete PostGIS Extension from GaussDB(DWS):

```
DROP EXTENSION postgis [CASCADE];
```

NOTE

If PostGIS Extension is the dependee of other objects (for example, geometry tables), you need to add the **CASCADE** keyword to delete all these objects.

11.3 PostGIS Support and Constraints

Supported Data Types

In GaussDB(DWS), PostGIS Extension support the following data types:

- box2d
- box3d
- geometry_dump
- geometry
- geography
- raster

NOTE

- The grid-related GUC parameters **postgis.gdal_datapath**, **postgis.gdal_enabled_drivers**, and **postgis.enable_outdb_rasters** have been enabled. You do not need to manually set them.
- If PostGIS is used by a user other than the creator of the PostGIS, set the following GUC parameters:
SET behavior_compat_options = 'bind_procedure_searchpath';

Supported Operators and Functions

Table 11-1 Operators and functions supported by PostGIS

Category	Function
Management functions	AddGeometryColumn, DropGeometryColumn, DropGeometryTable, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Liblwgeom_Version, PostGIS_Lib_Build_Date, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_Scripts_Build_Date, PostGIS_Scripts_Installed, PostGIS_Version, PostGIS_LibXML_Version, PostGIS_Scripts_Released, Populate_Geometry_Columns, UpdateGeometrySRID
Geometry constructors	ST_BdPolyFromText, ST_BdMPolyFromText, ST_Box2dFromGeoHash, ST_GeogFromText, ST_GeographyFromText, ST_GeogFromWKB, ST_GeomCollFromText, ST_GeomFromEWKB, ST_GeomFromEWKT, ST_GeometryFromText, ST_GeomFromGeoHash, ST_GeomFromGML, ST_GeomFromGeoJSON, ST_GeomFromKML, ST_GMLToSQL, ST_GeomFromText, ST_GeomFromWKB, ST_LineFromMultiPoint, ST_LineFromText, ST_LineFromWKB, ST_LinestringFromWKB, ST_MakeBox2D, ST_3DMakeBox, ST_MakeEnvelope, ST_MakePolygon, ST_MakePoint, ST_MakePointM, ST_MLineFromText, ST_MPointFromText, ST_MPolyFromText, ST_Point, ST_PointFromGeoHash, ST_PointFromText, ST_PointFromWKB, ST_Polygon, ST_PolygonFromText, ST_WKBToSQL, ST_WKTToSQL
Geometry accessors	GeometryType, ST_Boundary, ST_CoordDim, ST_Dimension, ST_EndPoint, ST_Envelope, ST_ExteriorRing, ST_GeometryN, ST_GeometryType, ST_InteriorRingN, ST_IsClosed, ST_IsCollection, ST_IsEmpty, ST_IsRing, ST_IsSimple, ST_IsValid, ST_IsValidReason, ST_IsValidDetail, ST_M, ST_NDims, ST_NPoints, ST_NRings, ST_NumGeometries, ST_NumInteriorRings, ST_NumInteriorRing, ST_NumPatches, ST_NumPoints, ST_PatchN, ST_PointN, ST_SRID, ST_StartPoint, ST_Summary, ST_X, ST_XMax, ST_XMin, ST_Y, ST_YMax, ST_YMin, ST_Z, ST_ZMax, ST_Zmflag, ST_ZMin
Geometry editors	ST_AddPoint, ST_Affine, ST_Force2D, ST_Force3D, ST_Force3DZ, ST_Force3DM, ST_Force4D, ST_ForceCollection, ST_ForceSFS, ST_ForceRHR, ST_LineMerge, ST_CollectionExtract, ST_CollectionHomogenize, ST_Multi, ST_RemovePoint, ST_Reverse, ST_Rotate, ST_RotateX, ST_RotateY, ST_RotateZ, ST_Scale, ST_Segmentize, ST_SetPoint, ST_SetSRID, ST_SnapToGrid, ST_Snap, ST_Transform, ST_Translate, ST_TransScale
Geometry outputs	ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_AsGeoJSON, ST_AsGML, ST_AsHEXEWKB, ST_AsKML, ST_AsLatLonText, ST_AsSVG, ST_AsText, ST_AsX3D, ST_GeoHash

Category	Function
Operators	&&, &&&, &<, &< , &>, <<, << , =, >>, @, &>, >>, ~, ~=, <->, <#>
Spatial relationships and measurements	ST_3DClosestPoint, ST_3DDistance, ST_3DDWithin, ST_3DDFullyWithin, ST_3DIntersects, ST_3DLongestLine, ST_3DMaxDistance, ST_3DShortestLine, ST_Area, ST_Azimuth, ST_Centroid, ST_ClosestPoint, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_LineCrossingDirection, ST_Disjoint, ST_Distance, ST_HausdorffDistance, ST_MaxDistance, ST_DistanceSphere, ST_DistanceSpheroid, ST_DFullyWithin, ST_DWithin, ST_Equals, ST_HasArc, ST_Intersects, ST_Length, ST_Length2D, ST_3DLength, ST_Length_Spheroid, ST_Length2D_Spheroid, ST_3DLength_Spheroid, ST_LongestLine, ST_OrderingEquals, ST_Overlaps, ST_Perimeter, ST_Perimeter2D, ST_3DPerimeter, ST_PointOnSurface, ST_Project, ST_Relate, ST_RelateMatch, ST_ShortestLine, ST_Touches, ST_Within
Geometry processing	ST_Buffer, ST_BuildArea, ST_Collect, ST_ConcaveHull, ST_ConvexHull, ST_CurveToLine, ST_DelaunayTriangles, ST_Difference, ST_Dump, ST_DumpPoints, ST_DumpRings, ST_FlipCoordinates, ST_Intersection, ST_LineToCurve, ST_MakeValid, ST_MemUnion, ST_MinimumBoundingCircle, ST_Polygonize, ST_Node, ST_OffsetCurve, ST_RemoveRepeatedPoints, ST_SharedPaths, ST_Shift_Longitude, ST_Simplify, ST_SimplifyPreserveTopology, ST_Split, ST_SymDifference, ST_Union, ST_UnaryUnion
Linear referencing	ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineSubstring, ST_LocateAlong, ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint, ST_AddMeasure
Miscellaneous functions	ST_Accum, Box2D, Box3D, ST_Expand, ST_Extent, ST_3DExtent, Find_SRID, ST_MemSize
Exceptional functions	PostGIS_AddBBox, PostGIS_DropBBox, PostGIS_HasBBox
Raster Management Functions	AddRasterConstraints, DropRasterConstraints, AddOverviewConstraints, DropOverviewConstraints, PostGIS_GDAL_Version, PostGIS_Raster_Lib_Build_Date, PostGIS_Raster_Lib_Version, and ST_GDALDrivers, and UpdateRasterSRID
Raster Constructors	ST_AddBand, ST_AsRaster, ST_Band, ST_MakeEmptyRaster, ST_Tile, and ST_FromGDALRaster

Category	Function
Raster Accessors	ST_GeoReference, ST_Height, ST_IsEmpty, ST_MetaData, ST_NumBands, ST_PixelHeight, ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_RasterToWorldCoord, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_Rotation, ST_SkewX, ST_SkewY, ST_SRID, ST_Summary, ST_UpperLeftX, ST_UpperLeftY, ST_Width, ST_WorldToRasterCoord, ST_WorldToRasterCoordX, ST_WorldToRasterCoordY
Raster Band Accessors	ST_BandMetaData, ST_BandNoDataValue, ST_BandIsNoData, ST_BandPath, ST_BandPixelType, and ST_HasNoBand
Raster Pixel Accessors and Setters	ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Value, ST_NearestValue, ST_Neighborhood, ST_SetValue, ST_SetValues, ST_DumpValues, and ST_PixelOfValue
Raster Editors	ST_SetGeoReference, ST_SetRotation, ST_SetScale, ST_SetSkew, ST_SetSRID, ST_SetUpperLeft, ST_Resample, ST_Rescale, ST_Reskew, and ST_SnapToGrid, ST_Resize, and ST_Transform
Raster Band Editors	ST_SetBandNoDataValue and ST_SetBandIsNoData
Raster Band Statistics and Analytics	ST_Count, ST_CountAgg, ST_Histogram, ST_Quantile, ST_SummaryStats, ST_SummaryStatsAgg, and ST_ValueCount
Raster Outputs	ST_AsBinary, ST_AsGDALRaster, ST_AsJPEG, ST_AsPNG, and ST_AsTIFF
Raster Processing	ST_Clip, ST_ColorMap, ST_Intersection, ST_MapAlgebra, ST_Reclass, and ST_Union ST_Distinct4ma, ST_InvDistWeight4ma, ST_Max4ma, ST_Mean4ma, ST_Min4ma, ST_MinDist4ma, ST_Range4ma, ST_StdDev4ma, and ST_Sum4ma, ST_Aspect, ST_HillShade, ST_Roughness, ST_Slope, ST_TPI, ST_TRI, Box3D, ST_ConvexHull, ST_DumpAsPolygons, and ST_Envelope, ST_MinConvexHull, ST_Polygon, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Disjoint, ST_Intersects, and ST_Overlaps, ST_Touches, ST_SameAlignment, ST_NotSameAlignmentReason, ST_Within, ST_DWithin, and ST_DFullyWithin
Raster Operators	&&, &<, &>, =, @, ~=, and ~

Spatial Indexes

In GaussDB(DWS), PostGIS Extension supports Generalized Search Tree (GIST) spatial indexes. This index type is inapplicable to partitioned tables. Different from B-tree indexes, GIS indexes are adaptable to all kinds of irregular data structures,

which can effectively improve the retrieval efficiency for geometry and geographic data.

Run the following command to create a GiST index:

```
CREATE INDEX indexname ON tablename USING GIST ( geometryfield );
```

Extension Constraints

- Only row-store tables are supported.
- Only Oracle-compatible databases are supported.
- The topology object management module, Topology, is not supported.
- BRIN indexes are not supported.
- The **spatial_ref_sys** table can only be queried during scale-out.

11.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS)

This document contains open source software notice for the product. And this document is confidential information of copyright holder. Recipient shall protect it in due care and shall not disseminate it without permission.

Warranty Disclaimer

This document is provided "as is" without any warranty whatsoever, including the accuracy or comprehensiveness. Copyright holder of this document may change the contents of this document at any time without prior notice, and copyright holder disclaims any liability in relation to recipient's use of this document.

Open source software is provided by the author "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of open source software, even if advised of the possibility of such damage.

Copyright Notice And License Texts

Software: postgis-2.4.2

Copyright notice:

"Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Copyright 2008 Kevin Neufeld

Copyright (c) 2009 Walter Bruce Sinclair

Copyright 2006-2013 Stephen Woodbridge.

Copyright (c) 2008 Walter Bruce Sinclair
Copyright (c) 2012 TJ Holowaychuk <tj@vision-media.ca>
Copyright (c) 2008, by Attractive Chaos <attractivechaos@aol.co.uk>
Copyright (c) 2001-2012 Walter Bruce Sinclair
Copyright (c) 2010 Walter Bruce Sinclair
Copyright 2006 Stephen Woodbridge
Copyright 2006-2010 Stephen Woodbridge.
Copyright (c) 2006-2014 Stephen Woodbridge.
Copyright (c) 2017, Even Rouault <even.rouault at spatialys.com>
Copyright (C) 2004-2015 Sandro Santilli <strk@kbt.io>
Copyright (C) 2008-2011 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright 2015 Nicklas Avén <nicklas.aven@jordogskog.no>
Copyright 2008 Paul Ramsey
Copyright (C) 2012 Sandro Santilli <strk@kbt.io>
Copyright 2012 Sandro Santilli <strk@kbt.io>
Copyright (C) 2014 Sandro Santilli <strk@kbt.io>
Copyright 2013 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2009 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2008 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2011 Sandro Santilli <strk@kbt.io>
Copyright 2015 Daniel Baston
Copyright 2009 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2014 Kashif Rasul <kashif.rasul@gmail.com> and
Shoaib Burq <saburq@gmail.com>
Copyright 2013 Sandro Santilli <strk@kbt.io>
Copyright 2010 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2015 Sandro Santilli <strk@kbt.io>
Copyright (C) 2009 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011 Sandro Santilli <strk@kbt.io>
Copyright 2010 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2014 Nicklas Avén

Copyright 2011-2016 Regina Obe
Copyright (C) 2008 Paul Ramsey
Copyright (C) 2011-2015 Sandro Santilli <strk@kbt.io>
Copyright 2010-2012 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright (C) 2015 Daniel Baston <dbaston@gmail.com>
Copyright (C) 2013 Nicklas Avén
Copyright (C) 2016 Sandro Santilli <strk@kbt.io>
Copyright 2017 Darafei Praliaskouski <me@komzpa.net>
Copyright (c) 2016, Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011-2012 Sandro Santilli <strk@kbt.io>
Copyright (C) 2011 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2007-2008 Mark Cave-Ayland
Copyright (C) 2001-2006 Refrations Research Inc.
Copyright 2015 Daniel Baston <dbaston@gmail.com>
Copyright 2009 David Skea <David.Skea@gov.bc.ca>
Copyright (C) 2012-2015 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2012-2015 Sandro Santilli <strk@kbt.io>
Copyright 2001-2006 Refrations Research Inc.
Copyright (C) 2004 Refrations Research Inc.
Copyright 2011-2014 Sandro Santilli <strk@kbt.io>
Copyright 2009-2010 Sandro Santilli <strk@kbt.io>
Copyright 2015-2016 Daniel Baston <dbaston@gmail.com>
Copyright 2011-2015 Sandro Santilli <strk@kbt.io>
Copyright 2007-2008 Mark Cave-Ayland
Copyright 2012-2013 Oslandia <infos@oslandia.com>
Copyright (C) 2015-2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2001-2003 Refrations Research Inc.
Copyright 2016 Sandro Santilli <strk@kbt.io>
Copyright 2011 Kashif Rasul <kashif.rasul@gmail.com>
Copyright (C) 2014 Nicklas Avén
Copyright (C) 2010 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2010-2015 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011 Sandro Santilli <strk@kbt.io>

Copyright (C) 2011-2014 Sandro Santilli <strk@kbt.io>
Copyright (C) 1984, 1989-1990, 2000-2015 Free Software Foundation, Inc.
Copyright (C) 2011 Paul Ramsey
Copyright 2001-2003 Refrations Research Inc.
Copyright 2009-2010 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2010-2012 Oslandia
Copyright 2006 Corporacion Autonoma Regional de Santander
Copyright 2013 Nicklas Avén
Copyright 2011-2016 Arrival 3D, Regina Obe
Copyright (C) 2009 David Skea <David.Skea@gov.bc.ca>
Copyright (C) 2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2009-2012 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2010 - Oslandia
Copyright (C) 2006 Mark Leslie <mark.leslie@lisasoft.com>
Copyright (C) 2008-2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (C) 2009-2015 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2010 Olivier Courtin <olivier.courtin@camptocamp.com>
Copyright 2010 Nicklas Avén
Copyright 2012 Paul Ramsey
Copyright 2011 Nicklas Avén
Copyright 2002 Thamer Alharbash
Copyright 2011 OSGeo
Copyright (C) 2009-2011 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (C) 2004-2007 Refrations Research Inc.
Copyright 2010 LISAssoft Pty Ltd
Copyright 2010 Mark Leslie
Copyright (c) 1999, Frank Warmerdam
Copyright 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (c) 2007, Frank Warmerdam
Copyright 2008 OpenGeo.org
Copyright (C) 2008 OpenGeo.org
Copyright (C) 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright 2010 LISAssoft
Copyright (C) 2010 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (c) 1999, 2001, Frank Warmerdam
Copyright (C) 2016-2017 Bjørn Harrell <bjorn@wololo.org>
Copyright (C) 2017 Danny Goette <danny.goette@fem.tu-ilmenau.de>
Copyright 2009-2011 Paul Ramsey <pramsey@cleverelephant.ca>
^copyright^
Copyright 2012 (C) Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2006 Refrations Research Inc.
Copyright 2009 Paul Ramsey <pramsey@opengeo.org>
Copyright 2001-2009 Refrations Research Inc.
Copyright (C) 2010 Olivier Courtin <olivier.courtin@oslandia.com>
By Nathan Wagner, copyright disclaimed,
this entire file is in the public domain
Copyright 2009-2011 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright (C) 2001-2005 Refrations Research Inc.
Copyright 2001-2011 Refrations Research Inc.
Copyright 2009-2014 Sandro Santilli <strk@kbt.io>
Copyright (C) 2008 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2010 Sandro Santilli <strk@kbt.io>
Copyright 2012 J Smith <dark.panda@gmail.com>
Copyright 2009 - 2010 Oslandia
Copyright 2009 Oslandia
Copyright 2001-2005 Refrations Research Inc.
Copyright 2016 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2016 Daniel Baston <dbaston@gmail.com>
Copyright (C) 2011 OpenGeo.org
Copyright (c) 2003-2017, Troy D. Hanson <http://troydhanson.github.com/uthash/>
Copyright (C) 2011 Regents of the University of California
Copyright (C) 2011-2013 Regents of the University of California
Copyright (C) 2010-2011 Jorge Arevalo <jorge.arevalo@deimos-space.com>
Copyright (C) 2010-2011 David Zwarg <dzwarg@azavea.com>

Copyright (C) 2009-2011 Pierre Racine <pierre.racine@sbf.ulaval.ca>
Copyright (C) 2009-2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2008-2009 Sandro Santilli <strk@kbt.io>
Copyright (C) 2013 Nathaneil Hunter Clay <clay.nathaniel@gmail.com>
Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>
Copyright (C) 2013 Bborie Park <dustymugs@gmail.com>
Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>
(C) 2009 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009-2010 Jorge Arevalo <jorge.arevalo@deimos-space.com>
Copyright (C) 2012 Regents of the University of California
Copyright (C) 2013 Regents of the University of California
Copyright (C) 2012-2013 Regents of the University of California
Copyright (C) 2009 Sandro Santilli <strk@kbt.io>
"

License: The GPL v2 License.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software

or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.?

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all

derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Software:Geos

Copyright notice:

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2013 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refrations Research

Copyright 2004 Sean Gillies, sgillies@frie.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Niall Dawson <niall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public

Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on,

the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and

is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in

non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The

former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under

copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the

Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1

above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in

themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or

collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you

can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany

it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to

distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a

work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License.

Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object

file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work

under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system,

rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception,

the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on

which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot

use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the

integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time.

Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is

copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status

of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990

Ty Coon, President of Vice

That's all there is to it!

Software: JSON-C

Copyright notice:

Copyright (c) 2004, 2005 Metaparadigm Pte. Ltd.

Copyright (c) 2009-2012 Eric Haszlakiewicz

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Copyright (c) 2009 Hewlett-Packard Development Company, L.P.

Copyright 2011, John Resig

Copyright 2011, The Dojo Foundation

Copyright (c) 2012 Eric Haszlakiewicz

Copyright (c) 2009-2012 Hewlett-Packard Development Company, L.P.

Copyright (c) 2008-2009 Yahoo! Inc. All rights reserved.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) 2013 Metaparadigm Pte. Ltd.

License: MIT License

Copyright (c) 2009-2012 Eric Haszlakiewicz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Software: proj

Copyright notice:

"Copyright (C) 2010 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2007 Douglas Gregor <doug.gregor@gmail.com>
Copyright (C) 2007 Troy Straszheim
CMake, Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>)
Copyright (C) 2011 Nicolas David <nicolas.david@ign.fr>
Copyright (c) 2000, Frank Warmerdam
Copyright (c) 2011, Open Geospatial Consortium, Inc.
Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,
2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.
Copyright (c) Charles Karney (2012-2015) <charles@karney.com> and licensed
Copyright (c) 2005, Antonello Andrea
Copyright (c) 2010, Frank Warmerdam
Copyright (c) 1995, Gerald Evenden
Copyright (c) 2000, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2010, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2013, Frank Warmerdam
Copyright (c) 2003 Gerald I. Evenden
Copyright (c) 2012, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2002, Frank Warmerdam
Copyright (c) 2004 Gerald I. Evenden
Copyright (c) 2012 Martin Raspaud
Copyright (c) 2001, Thomas Flemming, tf@ttqv.com
Copyright (c) 2002, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2009, Frank Warmerdam
Copyright (c) 2003, 2006 Gerald I. Evenden
Copyright (c) 2011, 2012 Martin Lambers <marlam@marlam.de>
Copyright (c) 2006, Andrey Kiselev
Copyright (c) 2008-2012, Even Rouault <even dot rouault at mines-paris dot org>
Copyright (c) 2001, Frank Warmerdam
Copyright (c) 2001, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2008 Gerald I. Evenden
"

License: MIT License
Please see above

Software: libxml2

Copyright notice:

"See Copyright for the status of this software.

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Copyright (C) 2003 Daniel Veillard.

copy: see Copyright for the status of this software.

copy: see Copyright for the status of this software

copy: see Copyright for the status of this software.

Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

Copy: See Copyright for the status of this software.

See COPYRIGHT for the status of this software

Copyright (C) 2000 Gary Pennington and Daniel Veillard.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,
2007 Free Software Foundation, Inc.

Copyright (C) 1998 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese <breese@users.sourceforge.net>

Copyright (C) 2000 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese and Daniel Stenberg.

See Copyright for the status of this software

"

License: MIT License

Please see above

12 Real-Time GaussDB(DWS)

12.1 Time Series Engine

12.1.1 Overview

The real-time GaussDB(DWS) provides a self-developed time series engine, which provides the time series computing capability based on column-store tables through time series functions, partition management, and other functions.

NOTE

- Real-time GaussDB(DWS) does not support SSL connections.
- Real-time GaussDB(DWS) does not MRS data source connections.

12.1.2 Usage

12.1.2.1 Partition Management

TSDB extension engine provides scheduled policy APIs for adding and deleting partitions. You can add and delete policies for existing partitioned tables as needed. If you create a times series table, a partition management policy exists. In this case, do not add a partition policy again. If you need to modify the partition management policy, you are advised to use the time series DDL statement **ALTER TABLE**.

Permission check is performed on the partition management interface. Only the table owner and administrator have the permission to invoke the interface.

API Details

- Determining whether the time series engine is available.

check_engine_enable

Example:

```
SELECT check_engine_enable();  
check_engine_enable
```



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

- Check whether the time series GUC switch is enabled.

is_tsdb_enable

Example:

```
SELECT is_tsdb_enable();  
is_tsdb_enable  
-----  
t  
(1 row)
```

- Add a partition creation policy.

add_create_partition_policy

Table 12-1 Parameter description

Parameter	Type	Description	Mandatory or Not
relationname	name	Partitioned table name	Yes
boundaries_interval	interval	Partition interval. Range: 1 hour~100 year	Yes
schedule_interval	interval	Policy execution interval. Range: 1 hour~100 year	Yes

Example:

```
SELECT add_create_partition_policy('example', interval '1 day', interval '1 day');
```

- Delete a partition creation policy.

remove_create_partition_policy

Table 12-2 Parameter description

Parameter	Type	Description	Mandatory or Not
relationname	name	Partitioned table name	Yes

Example:

```
SELECT remove_create_partition_policy('example');
```

- Create a partition deletion policy.

add_drop_partition_policy

Table 12-3 Parameter description

Parameter	Type	Description	Mandatory or Not
relationname	name	Partitioned table name	Yes
older_than	interval	Data retention period	Yes
schedule_interval	interval	Policy execution interval. Range: 1 hour~100 year	No

Example:

```
SELECT add_drop_partition_policy('exmaple', interval '7 days');
```

- Delete a partition deletion policy.
remove_drop_partition_policy

Table 12-4 Parameter description

Parameter	Type	Description	Mandatory or Not
relationname	name	Partitioned table name	Yes

Example:

```
SELECT remove_drop_partition_policy('exmaple');
```

12.1.2.2 Time Series Function

The TSDB engine provides basic computing capabilities in time series scenarios through time series calculation functions.

Time Series Calculation Functions

Table 12-5 Functions supported by time series calculation

Description	Function
Supplements the missing time values in the time window in a group. By default, values are sorted by time.	time_fill()
Performs interpolation on the missing aggregate value in the time window in a group by using the most recently seen value.	fill_last()

Description	Function
Performs modulo operation on the timestamp type based on any time window size.	time_window()
Sorts a given column by time series and returns the first value.	first()
Sorts a given column by time series and returns the last value.	last()
Sorts a given column by time series and returns a percentile value.	percentile_cont()
Performs interpolation on the data that is empty in each group.	fill()
Sorts a given column by time series and returns the first <i>K</i> tuples.	top_k()
Sorts a given column by time series and returns the last <i>K</i> tuples.	bottom_k()
Calculates the difference between two rows sorted by time.	delta()
Calculates the difference between the maximum value and the minimum value in a specified period.	spread()

time_fill(window_width interval, time timestamp(tz), start timestamp(tz), finish timestamp(tz))

This function is used to supplement the missing time windows in a group to make data complete. The supplemented values are sorted by time by default.

Table 12-6 Parameter description

Parameter	Type	Description	Required/Optional
window_width	interval	Interval of each time window	Required
time	timestamp	Time column	Required
start	timestamp	Start time	Required
finish	timestamp	End time	Required

 NOTE

- The unit of **window_width** can be microsecond, millisecond, second, min, hour, day, week, or a positive number. The default unit is day.
- The **time_fill** function needs to be used as an aggregation function. The **GROUP BY** clause needs to reference its own calculation result and cannot be nested with itself. The clause cannot be called multiple times in a single query, used as a lower-layer computing node, or used with the **WITHIN GROUP** clause.
- The **start** timestamp must be smaller than the **finish** timestamp, and their interval must be greater than the value of **window_width**.
- All parameters cannot be null. Values of **start** and **finish** must be specified.

Example:

```
SELECT
  time_fill(interval '1 min', time,
    '2019-07-11 23:59:01'::timestampz,
    '2019-07-12 00:03:01'::timestampz) as fill_time
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz),
    ('2019-07-12 00:00:00'::timestampz)) v(time)
GROUP BY fill_time;
```

fill_last(value anyelement)

This function is used to supplement the aggregate value corresponding to the missing time window in a group. The supplement policy is to use the aggregate value of the previous time window. If the previous time window is empty, no information is supplemented.

Table 12-7 Parameter description

Parameter	Type	Description	Required/Optional
value	anyelement	Value to be supplemented forward	Required

 NOTE

- **fill_last** and **time_fill** must be used together.
- This function cannot be used together with window functions or the **WITHIN GROUP** clause.

Example:

```
SELECT
  time_fill(interval '1 min', time,
    '2019-07-11 23:59:01'::timestampz,
    '2019-07-12 00:03:01'::timestampz) as fill_time,
  fill_last(min(value))
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz,1),
    ('2019-07-12 00:01:00'::timestampz,3)) v(time,value)
GROUP BY fill_time,value;
```

time_window(window_width interval, time timestamp(tz), origin timestamp(tz))

This function is used to perform modulo operation on the timestamp type based on any time window size.

Table 12-8 Parameter description

Parameter	Type	Description	Required/Optional
window_width	interval	Time window	Required
time	timestamp(tz)	Timestamp column	Required
origin	timestamp(tz)	Original timestamp	Optional

 NOTE

- The unit of **window_width** can be microsecond, millisecond, second, min, hour, day, week, or a positive number. The default unit is day.
- The values of **window_width** and **time** must be of the correct type and cannot be empty.

Example:

```
SELECT
  time_window(interval '1 min', time),
  time_window(interval '3 min', time, '2019-07-11 00:02:05'::timestamptz)
FROM
  (VALUES ('2019-07-12 00:00:00'::timestamptz),
   ('2019-07-12 00:01:01'::timestamptz)) v(time);
```

first(value_array anyarray, time_array timearray)

This function is used to sort a given column by time series and return the first value.

Table 12-9 Parameter description

Parameter	Type	Description	Required/Optional
value_array	anyarray	Array of queried column	Required
time_array	timearray	Array of time values	Required

 NOTE

This function cannot be directly used for a large amount of data.

Example:

```
SELECT
  first(array_agg(value), array_agg(time))
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz,'c','a'),
    ('2019-07-12 00:01:00'::timestampz,'b','a'),
    ('2019-07-12 00:02:01'::timestampz,'a','b')) v(time, value, tag)
GROUP BY tag;
```

last(value_array anyarray, time_array timearray)

This function is used to sort a given column by time series and return the last value.

Table 12-10 Parameter description

Parameter	Type	Description	Required/ Optional
value_array	anyarray	Array of queried column	Required
time_array	timearray	Array of time values	Required

NOTE

This function cannot be directly used for a large amount of data.

Example:

```
SELECT
  last(array_agg(value), array_agg(time))
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz,'c','a'),
    ('2019-07-12 00:01:00'::timestampz,'b','a'),
    ('2019-07-12 00:02:01'::timestampz,'a','b')) v(time, value, tag)
GROUP BY tag;
```

percentile_cont(percentile float)

This function is used to sort a given column by time series and return a percentile value.

Table 12-11 Parameter description

Parameter	Type	Description	Required/ Optional
percentile	float	Specifies the percentile. The value ranges from 0 to 1. The precision value is of the floating point type. For example, 0.95 means the 95 th percentile value.	Required

 NOTE

- **percentile** is a decimal from 0 to 1. The value is of the floating point type and does not support the percent sign (%).
- This parameter must be used together with **WITHIN GROUP (ORDER BY)** to specify the column to be calculated. The column must be of the numeric type.

Example:

```
SELECT
  percentile_cont(0.5) WITHIN GROUP (ORDER BY value)
FROM
  (VALUES ('2019-07-12 00:00:01'::timestampz,'red',1),
   ('2019-07-12 00:00:01'::timestampz,'blue',2)) v(time,color,value);
```

fill(value anyelement)

This function is used to perform interpolation on the data that is empty in each group. Different from **time_fill** which is used to supplement the missing time values, **fill_last** is used to supplement only other values of the missing time, and **fill** is used to supplement the null value in the existing data.

Table 12-12 Parameter description

Parameter	Type	Description	Required/Optional
Value	anyelement	Value to be supplemented	Required

 NOTE

The **fill** function must be used together with the **over** window function.

Example:

```
SELECT
  time,
  CASE WHEN value is not null THEN value else fill(value) over(partition by time) END
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz, 1),
   ('2019-07-12 00:00:00'::timestampz, NULL),
   ('2019-07-12 00:02:00'::timestampz, 3)) v(time,value);
```

top_k(array real[],k int)

This function is used to sort a given column by time series and return the first K tuples.

Table 12-13 Parameter description

Parameter	Type	Description	Required/Optional
array	real[]	Array to be input	Required
k	int	Specified first K tuples	Required

 NOTE

- The value of **k** is a non-negative number smaller than the value of **array**.
- **array** cannot be empty.
- This function cannot be directly used for a large amount of data.

Example:

```
SELECT
  top_k(array_agg(v), 0.5),v2
FROM
  (VALUES (1,'a'),(2,'a'),(3,'a'),(4,'a'),(5,'a'),(6,'a')) v(v,v2)
GROUP BY v2;
```

bottom_k(array real[],k int)

This function is used to sort a given column by time series and return the last *K* tuples.

Table 12-14 Parameter description

Parameter	Type	Description	Required/ Optional
array	real[]	Array to be input	Required
k	int	Specified last <i>K</i> tuples	Required

 NOTE

- The value of **k** is a non-negative number smaller than the value of **array**.
- **array** cannot be empty.
- This function cannot be directly used for a large amount of data.

Example:

```
SELECT
  bottom_k(array_agg(v), 0.5),v2
FROM
  (VALUES (1,'a'),(2,'a'),(3,'a'),(4,'a'),(5,'a'),(6,'a')) v(v,v2)
GROUP BY v2;
```

delta(field numeric) over(rows 1 preceding)

Calculates the difference between two rows sorted by time.

Table 12-15 Parameter description

Parameter	Type	Description	Required/ Optional
field	Numeric	Column to be calculated	Required

 NOTE

- This function is used to calculate the interpolation between two adjacent rows sorted by time to monitor indicators such as traffic and speed.
- This function must be used together with the window function **OVER(ROWS 1 PRECEDING)**. However, do not use the function together with **OVER (PARTITION BY TAG ROWS 1 PRECEDING)** or **GROUP BY**. Otherwise, the result may not be desirable.

Example:

```
SELECT
    delta(value) over (rows 1 preceding)
FROM
    (VALUES ('2019-07-12 00:00:00'::timestampz, 1),('2019-07-12 00:01:00'::timestampz, 2),('2019-07-12
00:02:00'::timestampz, 3)) v(time,value);
```

spread(field numeric)

Calculates the difference between the maximum value and the minimum value in a specified period.

Table 12-16 Parameter description

Parameter	Type	Description	Required/ Optional
field	Numeric	Column to be calculated	Required

 NOTE

- This function is used to calculate the increment of each counter sorted by time.
- If there are less than two tuples in each group, the returned result is **0**. Do not use this function with the **OVER** window function.

Example:

```
SELECT
    SPREAD(value)
FROM
    (VALUES ('2019-07-12 00:00:00'::timestampz, 1),('2019-07-12 00:01:00'::timestampz, 2),('2019-07-12
00:02:00'::timestampz, 3)) v(time,value);
```

12.1.3 Support and Constraints

Extension Constraints

- The timing engine supports only some relational syntax. The details are as follows:

Syntax	Supported or Not
CREATE TABLE	Supported
DROP TABLE	Supported
INSERT	Supported
COPY	Supported

SELECT	Supported
TRUNCATE	Supported
EXPLAIN	Supported
ANALYZE	Supported
VACUUM	Supported
ALTER TABLE ADD(DROP) COLUMN	Supported
ALTER TABLE ADD(DROP) PARTITION	Supported
ALTER TABLE SET WITH OPTION	Supported
ALTER TABLE TRUNCATE PARTITION	Supported
CREATE INDEX	Not supported
DROP INDEX	Not supported
DELETE	Not supported
ALTER TABLE	Not supported
ALTER INDEX	Not supported
MERGE	Not supported
SELECT INTO	Not supported
UPDATE	Not supported
CREATE TABLE AS	Not supported

- The tag column in a time series table supports only the text type, and the field column supports only the numeric type.
- After time series is enabled, the compaction function is enabled by default.
- Time series tables do not support online capacity expansion.
- Time series tables do not support table-level backup.

13 Setting GUC Parameters

13.1 Viewing Parameter Values

GaussDB(DWS) uses a set of default GUC parameters after it is installed. You can modify the parameters to better fit your service scenario and data volume.

Procedure

Step 1 View the parameter values of a database.

- Use the **SHOW** command.
 - To view a certain parameter, run the following command:
`SHOW server_version;`
server_version indicates the database version.
 - Run the following command to view values of all parameters:
`SHOW ALL;`
- Use the **pg_settings** view.
 - To view a certain parameter, run the following command:
`SELECT * FROM pg_settings WHERE NAME='server_version';`
 - Run the following command to view values of all parameters:
`SELECT * FROM pg_settings;`

----End

Example

Run the following command to view the server version:

```
SHOW server_version;  
server_version  
-----  
9.2.4  
(1 row)
```

13.2 Resetting Parameters

Context

GaussDB(DWS) provides multiple methods to set the GUC parameters of databases, users, or sessions.

- Parameter names are case-insensitive.
- A parameter value can be an integer, floating point value, string, Boolean value, or enumerated value.
 - 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 **pg_settings** system catalog.
- For parameters using units, specify their units during the setting, or 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 CN and DN parameters at a time, but cannot do the same to other parameters.

For details about the parameters, see [GUC Parameter Usage](#).

Setting GUC Parameters

GaussDB(DWS) provides the following GUC parameters. [Table 13-1](#) lists their types and setting methods.

Table 13-1 GUC parameters

Type	Description	Setting Method
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 commands.	Method by a database administrator in Table 13-2
USERSET	Common user parameter. It can be set by any user at any time.	Method in Table 13-2

You can modify the GUC parameters in GaussDB(DWS) using the methods listed in [Table 13-2](#).

Table 13-2 Setting GUC parameters

No.	Method
Method	Set parameters at database, user, or session levels. - Set a database-level parameter. ALTER DATABASE <i>dbname</i> SET <i>paraname</i> TO <i>value</i>; The setting takes effect in the next session. - Set a user-level parameter. ALTER USER <i>username</i> SET <i>paraname</i> TO <i>value</i>; The setting takes effect in the next session. - Set a session-level parameter. SET <i>paraname</i> TO <i>value</i>; Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.

Procedure

The following example shows how to set **explain_perf_mode**.

Step 1 View the value of **explain_perf_mode**.

```
SHOW explain_perf_mode;
explain_perf_mode
-----
normal
(1 row)
```

Step 2 Set **explain_perf_mode**.

Perform one of the following operations:

- Set a database-level parameter.
ALTER DATABASE **postgres** **SET** *explain_perf_mode* **TO** *pretty*;
The setting takes effect in the next session.
- Set a user-level parameter.
ALTER USER **dbadmin** **SET** *explain_perf_mode* **TO** *pretty*;
The setting takes effect in the next session.
- Set a session-level parameter.
SET *explain_perf_mode* **TO** *pretty*;

Step 3 Check whether the parameter is correctly set.

```
SHOW explain_perf_mode;
explain_perf_mode
-----
pretty
(1 row)
```

----End

13.3 GUC Parameter Usage

The database provides many operation parameters. Configuration of these parameters affects 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 database.
- If the allowed maximum value of a parameter is **INT_MAX**, it indicates the maximum parameter value varies by OS.
- If the allowed maximum value of a parameter is **DBL_MAX**, it indicates the maximum parameter value varies by OS.

13.4 Connection and Authentication

13.4.1 Connection Settings

This section describes parameters related to the connection mode between the client and server.

application_name

Parameter description: Specifies the name of the client program connecting to the database.

Type: USERSET

Value range: a string

Default value: `gsql`

connection_info

Parameter description: Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner. (This is an O&M parameter. Do not configure it by yourself.)

Type: USERSET

Value range: a string

Default value: an empty string

NOTE

- An empty string indicates that the driver connected to the database does not support automatic setting of the **connection_info** parameter or the parameter is not set by users in applications.
- The following is an example of the concatenated value of **connection_info**:

```
{"driver_name":"ODBC","driver_version":"(GaussDB A 8.0.0 build 62e7353e) compiled at 2019-06-26 14:56:09 commit 5361 last mr 9168 debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
```

driver_name and **driver_version** are displayed by default. Whether **driver_path** and **os_user** are displayed is determined by users.

13.4.2 Security and Authentication (postgresql.conf)

This section describes parameters about how to securely authenticate the client and server.

session_timeout

Parameter description: Specifies the longest duration with no operations after the connection to the server.

Type: USERSET

Value range: an integer ranging from 0 to 86400. The minimum unit is second (s). 0 means to disable the timeout.

Default value: 10 min

NOTICE

The gsql client of GaussDB(DWS) 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_renegotiation_limit

Parameter description: Specifies the traffic volume over the SSL-encrypted channel before the session key is renegotiated. The renegotiation traffic limitation 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 sent and received traffic.

Type: USERSET

NOTE

You are advised to retain the default value, that is, disable the renegotiation mechanism. You are not advised to use the **gs_guc** tool or other methods to set the **ssl_renegotiation_limit** parameter in the **postgresql.conf** file. The setting does not take effect.

Value range: an integer ranging from 0 to **INT_MAX**. The unit is KB. 0 indicates that the renegotiation mechanism is disabled.

Default value: 0

13.4.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

comm_no_delay

Parameter description: Specifies whether to use the **NO_DELAY** attribute of the communication library connection. Restart the cluster for the setting to take effect.

Type: USERSET

Value range: Boolean

Default value: off

NOTICE

If packet loss occurs because a large number of packets are received per second, set this parameter to **off** to reduce the total number of packets.

comm_debug_mode

Parameter description: Specifies the debug mode of the TCP proxy communication library or SCTP communication library, that is, whether to print logs about the communication layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed debug log of the communication library is printed.
- **off** indicates the detailed debug log of the communication library is 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 package is received.

Type: USERSET

Value range: an integer ranging from 0 to 20000. The unit is millisecond (ms). **0** indicates that automatic ACK triggering is disabled.

Default value: 2000

comm_timer_mode

Parameter description: Specifies the timer mode of the TCP proxy communication library or SCTP communication library, that is, whether to print timer logs in each phase of the communication layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed timer log of the communication library is printed.
- **off** indicates the detailed timer log of the communication library is not printed.

Default value: off

comm_stat_mode

Parameter description: Specifies the statistics mode of the TCP proxy communication library or SCTP communication library, that is, whether to print statistics about the communication layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the statistics log of the communication library is printed.
- **off** indicates the statistics log of the communication library is not printed.

Default value: off

13.5 Resource Consumption

13.5.1 Disk Space

This section describes the disk space parameter, which is used to set a limit on the disk space for storing files flushed to the disk.

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.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. -1 indicates no limit.

Default value: -1

temp_file_limit

Parameter description: Specifies the size of a single file spilled to disk if splitting is triggered in a session. The temporary file can be the one used by sorting or hash tables, or cursors in a session.

This is a session-level setting.

Type: SUSERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. -1 indicates no limit.

Default value: -1

NOTICE

This parameter does not apply to disk space occupied by temporary tablespaces used for executing SQL queries.

13.5.2 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 operations hold critical locks and should be complete as quickly as possible. In GaussDB(DWS), cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is calculated as follows and is the maximum value of the following calculation results:

- $\text{vacuum\_cost\_delay} * \text{accumulated\_balance} / \text{vacuum\_cost\_limit}$
 - $\text{vacuum\_cost\_delay} * 4$
-

Context

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 value other than 0.

vacuum_cost_delay

Parameter description: Specifies the length of time that the process will sleep when **vacuum_cost_limit** has been exceeded.

Type: USERSET

Value range: an integer ranging from 0 to 100. The unit is millisecond (ms). A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

Default value: 0

NOTICE

- On many systems, the effective resolution of sleep length is 10 ms. 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 set to a small value, such as 10 or 20 milliseconds. Adjusting vacuum's resource consumption is best done by changing other parameters.
-

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 page.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

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.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

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 I/Os required to flush the dirty block out to disk again.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: 20

vacuum_cost_limit

Parameter description: Specifies the cost limit. The cleanup process will sleep if this limit is exceeded.

Type: USERSET

Value range: an integer ranging from 1 to 10000. The unit is ms.

Default value: 200

13.5.3 Asynchronous I/O Operations

enable_adio_debug

Parameter description: Specifies whether O&M personnel are allowed to generate some ADIO logs to locate ADIO issues. This parameter is used only by developers. Common users are advised not to use it.

Type: SUSERSET

Value range: Boolean

- **on** or **true** indicates the log switch is enabled.
- **off** or **false** indicates the log switch is disabled.

Default value: off

enable_fast_allocate

Parameter description: Specifies whether the quick allocation switch of the disk space is enabled. This switch can be enabled only in the XFS file system.

Type: SUSERSET

Value range: Boolean

- **on** or **true** indicates that this function is enabled.
- **off** or **false** indicates that the function is disabled.

Default value: off

prefetch_quantity

Parameter description: Specifies the number of row-store prefetches using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: 32 MB

backwrite_quantity

Parameter description: Specifies the number of row-store writes using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: 8 MB

cstore_prefetch_quantity

Parameter description: Specifies the number of column-store prefetches using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 32 MB

cstore_backwrite_quantity

Parameter description: Specifies the number of column-store writes using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 8 MB

cstore_backwrite_max_threshold

Parameter description: Specifies the maximum number of column-store writes buffered in the database using the ADIO.

Type: USERSET

Value range: An integer. The value range is from 4096 to INT_MAX/2 and the unit is KB.

Default value: 2 GB

fast_extend_file_size

Parameter description: Specifies the disk size that the row-store pre-scales using the ADIO.

Type: SUSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 8MB

effective_io_concurrency

Parameter description: Specifies the number of requests that can be simultaneously processed by the disk subsystem. For the RAID array, the parameter value must be the number of disk drive spindles in the array.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: 1

13.6 Parallel Data Import

GaussDB(DWS) 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 distinguish between the problems "the number of imported file records is empty" and "the imported file does not exist". If this parameter is set to **true** and the problem "the imported file does not exist" occurs, GaussDB(DWS) will report the error message "file does not exist".

Type: SUSET

Value range: Boolean

- **on** indicates 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 the messages of "the number of imported file records is empty" and "the imported file does not exist" are not distinguished when files are imported.

Default value: off

partition_mem_batch

Parameter description: Specifies the number of buffers. In order to optimize the inserting of column-store partitioned table in batches, the data is buffered during the inserting process and then written in the disk. If the value is too large, much memory will be consumed. If it is too small, the performance of inserting column-store partitioned tables in batches will deteriorate.

Type: USERSET

Value range: 1 to 65535

Default value: 256

partition_max_cache_size

Parameter description: Specifies the size of a data buffer. To optimize batch insertion to column-store partitioned tables, data is buffered during the insertion and then written into disks in batches. If the value is too large, much memory will be consumed. If it is too small, the performance of data insertion to column-store partitioned tables will deteriorate.

Type: USERSET

Value range:

- Column-store partitioned table: 4096 to INT_MAX/2. The unit is KB.

Default value: 2 GB

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 this way, 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.

Type: USERSET

Value range: Boolean

- **on** indicates that the GDS debug function is enabled.
- **off** indicates that the GDS debug function is disabled.

Default value: off

13.7 Write Ahead Logs

13.7.1 Settings

synchronous_commit

Parameter description: Specifies the synchronization mode of the current transaction.

Type: USERSET

Value range: enumerated values

- **on** indicates synchronization logs of a standby server are flushed to disks.
- **off** indicates asynchronous commit.
- **local** indicates local commit.

- **remote_write** indicates synchronization logs of a standby server are written to disks.
- **remote_receive** indicates synchronization logs of a standby server are required to receive data.

Default value: on

commit_delay

Parameter description: Specifies the duration of committed data be stored in the WAL buffer.

Type: USERSET

Value range: an integer, ranging from 0 to 100000 (unit: μ s). 0 indicates no delay.

Default value: 0

NOTICE

- When this parameter is set to a value other than 0, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then, the WALwriter 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 transactions are waiting to be submitted, the delay is a waste of time.

commit_siblings

Parameter description: Specifies a limit on the number of ongoing transactions. If the number of ongoing transactions is greater than the limit, a new transaction will wait for the period of time specified by [commit_delay](#) before it is submitted. If the number of ongoing transactions is less than the limit, the new transaction is immediately written into a WAL.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: 5

13.8 HA Replication

13.8.1 Primary Server

enable_mix_replication

Parameter description: Specifies how WAL logs and data are replicated between primary, standby, and secondary nodes.

Type: INTERNAL (fixed parameter, which can be viewed but not modified)

Value range: Boolean

- **on** indicates that the WAL log and data page mixed replication mode is enabled.
- **off** indicates that the WAL log and data page mixed replication mode is disabled.

Default value: off

NOTICE

This parameter is an INTERNAL parameter and cannot be modified in normal service scenarios. That is, the WAL log and data page mixed replication mode is disabled.

enable_data_replicate

Parameter description: Specifies the data synchronization mode between the primary and standby servers when data is imported to row-store tables in a database.

Type: USERSET

Value range: Boolean

- **on** indicates that data pages are used for the data synchronization between the primary and standby servers when data is imported to row-store tables in a database. This parameter cannot be set to **on** if **replication_type** is set to **1**.
- **off** indicates that the primary and standby servers synchronize data using Xlogs while the data is imported to a row-store table.

Default value: on

13.9 Query Planning

13.9.1 Optimizer Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways include adjusting the optimizer cost constants, manually running **ANALYZE**, increasing the value of the **default_statistics_target** configuration parameter, and adding the statistics collected in a specific column using **ALTER TABLE SET STATISTICS**.

enable_bitmapscan

Parameter description: Controls whether the query optimizer uses the bitmap-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the bitmap-scan plan type.
- **off** indicates the query optimizer does not use the bitmap-scan plan type.

Default value: on

enable_hashagg

Parameter description: Controls whether the query optimizer uses the Hash aggregation plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the Hash aggregation plan type.
- **off** indicates the query optimizer does not use the Hash aggregation plan type.

Default value: on

enable_hashjoin

Parameter description: Controls whether the query optimizer uses the Hash-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the Hash-join plan type.
- **off** indicates the query optimizer does not use the Hash-join plan type.

Default value: on

enable_indexscan

Parameter description: Controls whether the query optimizer uses the index-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the index-scan plan type.
- **off** indicates the query optimizer does not use the index-scan plan type.

Default value: on

enable_indexonlyscan

Parameter description: Controls whether the query optimizer uses the index-only-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the index-only-scan plan type.
- **off** indicates the query optimizer does not use the index-only-scan plan type.

Default value: on

enable_material

Parameter description: Controls whether the query optimizer uses materialization. It is impossible to suppress materialization entirely, but setting this parameter to **off** prevents the optimizer from inserting materialized nodes.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses materialization.
- **off** indicates the query optimizer does not use materialization.

Default value: on

enable_mergejoin

Parameter description: Controls whether the query optimizer uses the merge-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the merge-join plan type.
- **off** indicates the query optimizer does not use the merge-join plan type.

Default value: off

enable_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to fully scan internal tables. It is impossible to suppress nested-loop joins entirely, but setting this parameter to **off** allows the optimizer to choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the nested-loop join plan type.
- **off** indicates the query optimizer does not use the nested-loop join plan type.

Default value: off

enable_index_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to scan the parameterized indexes of internal tables.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the nested-loop join plan type.
 - **off** indicates the query optimizer does not use the nested-loop join plan type.
- Default value:** on

enable_seqscan

Parameter description: Controls whether the query optimizer uses the sequential scan plan type. It is impossible to suppress sequential scans entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the sequential scan plan type.
- **off** indicates the query optimizer does not use the sequential scan plan type.

Default value: on

enable_sort

Parameter description: Controls whether the query optimizer uses the sort method. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the sort method.
- **off** indicates the query optimizer does not use the sort method.

Default value: on

enable_tidscan

Parameter description: Controls whether the query optimizer uses the Tuple ID (TID) scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the TID scan plan type.
- **off** indicates the query optimizer does not use the TID scan plan type.

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.

Type: SUSET

Value range: Boolean

- **on** indicates the unlocking is allowed.
- **off** indicates the unlocking is not allowed.

Default value: off

enforce_oracle_behavior

Parameter description: Controls the rule matching modes of regular expressions.

Type: USERSET

Value range: Boolean

- **on** indicates that the ORACLE matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: on

enable_stream_concurrent_update

Parameter description: Controls the use of **stream** in concurrent updates. This parameter is restricted by the [enable_stream_operator](#) parameter.

Type: USERSET

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 whether the query optimizer uses streams.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses streams.
- **off** indicates the query optimizer does not use streams.

Default value: on

enable_stream_recursive

Parameter description: Specifies whether to push **WITH RECURSIVE** join queries to DN for processing.

Type: USERSET

Value range: Boolean

- **on:** **WITH RECURSIVE** join queries will be pushed down to DNs.
- **off:** **WITH RECURSIVE** join queries will not be pushed down to DNs.

Default value: on

max_recursive_times

Parameter description: Specifies the maximum number of **WITH RECURSIVE** iterations.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 200

enable_vector_engine

Parameter description: Controls whether the query optimizer uses the vectorized executor.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the vectorized executor.
- **off** indicates the query optimizer does not use the vectorized executor.

Default value: on

enable_broadcast

Parameter description: Controls whether the query optimizer uses the broadcast distribution method when it evaluates the cost of stream.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the broadcast distribution method.
- **off** indicates the query optimizer does not use the broadcast distribution method.

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.

Type: SUSERSET

Value range: Boolean

- **on** indicates the optimizer excludes internal table running costs when selecting the Hash Join cost path.

- **off** indicates the optimizer does not exclude internal table running costs when selecting the Hash Join cost path.

Default value: off

enable_fstream

Parameter description: Controls whether the query optimizer uses streams when it delivers statements. This parameter is only used for external HDFS tables.

This parameter has been discarded. To reserve forward compatibility, set this parameter to **on**, but the setting does not make a difference.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses streams.
- **off** indicates the query optimizer does not use streams.

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 query optimizer to generate which type of hashagg plans.

Type: USERSET

Value range: an integer ranging from 0 to 3.

- When the value is set to **1**, the first plan is forcibly generated.
- When the value 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 value 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 value is set to **0**, the query optimizer chooses the most 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 all of the columns are not in the distribution column, you need to select one **group by** column for redistribution. This parameter controls the policy of selecting a redistribution column.

Type: USERSET

Value range: Boolean

- **on** indicates the column that can be redistributed and evaluates the most distinct value for redistribution.
- **off** indicates the first column that can be redistributed for redistribution.

Default value: off

enable_valuepartition_pruning

Parameter description: Specifies whether the DFS partitioned table is dynamically or statically optimized.

Type: USERSET

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 computing 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 an appropriate one for the current operator.

Type: USERSET

Value range: a string

- **optimal:** The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the DNs in the Node Groups on which the current user has the COMPUTE permission.
- **query:** The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the 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 computing Node Groups consists of the Node Group 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 computing 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.

Type: USERSET

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 to compute.

Default value: off

stream_multiple

Parameter description: Specifies the weight used for optimizer to calculate the final cost of stream operators.

The base stream cost is multiplied by this weight to make the final cost.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 1

NOTICE

This parameter is applicable only to Redistribute and Broadcast streams.

qrw_inlist2join_optmode

Parameter description: Specifies whether enable inlist-to-join (inlist2join) query rewriting.

Type: USERSET

Value range: a string

- **disable:** inlist2join disabled
- **cost_base:** cost-based inlist2join query rewriting
- **rule_base:** forcible rule-based inlist2join query rewriting
- A positive integer: threshold of Inlist2join query rewriting. If the number of elements in the list is greater than the threshold, the rewriting is performed.

Default value: cost_base

skew_option

Parameter description: Specifies whether an optimization policy is used

Type: USERSET

Value range: a string

- **off:** policy disabled
- **normal:** radical policy. All possible skews are optimized.
- **lazy:** conservative policy. Uncertain skews are ignored.

Default value: normal

13.9.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described in this section are measured on an arbitrary scale. Only their relative values matter, therefore scaling them all in or out by the same factor will result in no differences 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.

Type: USERSET

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.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 4

NOTE

- Although the server allows you to set the value of **random_page_cost** to 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.
- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Comparing to **seq_page_cost**, reducing this value will cause the system to prefer index scans and raising it makes index scans relatively more expensive. You can increase or decrease both values at the same time to change the disk I/O cost relative to CPU cost.

cpu_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each row during a query.

Type: USERSET

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.

Type: USERSET

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 during a query.

Type: USERSET

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.

When setting this parameter you should consider both GaussDB(DWS)'s shared buffer and the kernel's disk cache. Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space.

This parameter has no effect on the size of shared memory allocated by GaussDB(DWS). It is used only for estimation purposes and does not reserve kernel disk cache. The value is in the unit of disk page. Usually the size of each page is 8192 bytes.

Type: USERSET

Value range: an integer ranging is from 1 to INT_MAX. The unit is 8 KB.

A value greater than the default one may enable index scanning, and a value less than the default one may enable sequence scanning.

Default value: 128 MB

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.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 0

13.9.3 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 cost of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

geqo

Parameter description: Controls the use of genetic query optimization.

Type: USERSET

Value range: Boolean

- **on** indicates GEQO is enabled.
- **off** indicates GEQO is disabled.

Default value: on

NOTICE

Generally, do not set this parameter to **off**. **geqo_threshold** provides more subtle control of GEQO.

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.

Type: USERSET

Value range: an integer ranging from 2 to INT_MAX

Default value: 12

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.
-

geqo_effort

Parameter description: Controls the trade-off between planning time and query plan quality in GEQO.

Type: USERSET

Value range: an integer ranging from 1 to 10

Default value: 5

NOTICE

- Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.
 - **geqo_effort** does not have direct effect. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. You can manually set other parameters as required.
-

geqo_pool_size

Parameter description: Specifies the pool size used by GEQO, that is, the number of individuals in the genetic population.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

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(DWS) selects a proper value based on **geqo_effort** and the number of tables.

Default value: 0

geqo_generations

Parameter description: Specifies the number parameter iterations of the algorithm used by GEQO.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

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.

Type: USERSET

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.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Varying the value changes the setting of join paths explored, and may result in a better or worse path being found.

Default value: 0

13.9.4 Other Optimizer Options

default_statistics_target

Parameter description: Specifies the default statistics target for table columns without a column-specific target set via **ALTER TABLE SET STATISTICS**. If this parameter is set to a positive number, it indicates the number of samples of statistics information. If this parameter is set to a negative number, percentage is used to set the statistic target. The negative number converts to its corresponding percentage, for example, -5 means 5%.

Type: USERSET

Value range: an integer ranging from -100 to 10000

NOTICE

- A larger positive number than the parameter 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 optimize the query plan. For details, see [Hint-based Tuning](#).
- If this parameter is set to a negative value, the number of samples is greater than or equal to 2% of the total data volume, and the number of records in user tables is less than 1.6 million, the time taken by running **ANALYZE** will be longer than when this parameter uses its default value.
- If this parameter is set to a negative value, the autoanalyze function is disabled.

Default value: 100

constraint_exclusion

Parameter description: Controls the query optimizer's use of table constraints to optimize queries.

Type: USERSET

Value range: enumerated values

- **on** indicates the constraints for all tables are examined.
- **off**: No constraints are examined.
- **partition** indicates that only constraints for inherited 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, this parameter is set to **on** by default to partition tables. If this parameter is set to **on**, 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.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Smaller values than the default value 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.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

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.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

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. **OPTIMIZE_PLAN** indicates the optimal plan, the cost of which is estimated using the dynamic planning algorithm, and its value is **0**. **RANDOM_PLAN** indicates the plan that is randomly generated. If **plan_mode_seed** is set to **-1**, you do not need to specify the value of the seed identifier. Instead, the optimizer generates a random integer ranging from **1** to **2147483647**, and then generates a random execution plan based on this random number. If **plan_mode_seed** is set to an integer ranging from **1** to **2147483647**, you need to specify the value of the seed identifier, and the optimizer generates a random execution plan based on the seed value.

Type: USERSET

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 different random execution plans, which may not be the optimal. 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_hdfs_predicate_pushdown

Parameter description: Specifies whether the function of pushing down predicates the native data layer is enabled.

Type: SUSET

Value range: Boolean

- **on** indicates this function is enabled.
- **off** indicates this function is disabled.

Default value: on

enable_random_datanode

Parameter description: Specifies whether the function that random query about DN in the replication table is enabled. A complete data table is stored on each DN for random retrieval to release the pressure on nodes.

Type: USERSET

Value range: Boolean

- **on:** This function is enabled.
- **off:** This function is disabled.

Default value: on

hashagg_table_size

Parameter description: Specifies the hash table size during the execution of the HASH JOIN operation.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX/2

Default value: 0

enable_codegen

Parameter description: Specifies whether code optimization can be enabled. Currently, the code optimization uses the LLVM optimization.

Type: USERSET

Value range: Boolean

- **on** indicates code optimization can be enabled.
- **off** indicates code optimization cannot be enabled.

NOTICE

Currently, the LLVM optimization only supports the vectorized executor and SQL on Hadoop features. You are advised to set this parameter to **off** in other cases.

Default value: on

codegen_strategy

Parameter description: Specifies the codegen optimization strategy that is used when an expression is converted to codegen-based.

Type: USERSET

Value range: enumerated values

- **partial** indicates that you can still call the LLVM dynamic optimization strategy using the codegen framework of an expression even if functions that are not codegen-based exist in the expression.
- **pure** indicates that the LLVM dynamic optimization strategy can be called only when all functions in an expression can be codegen-based.

NOTICE

In the scenario where query performance reduces after the codegen function is enabled, you can set this parameter to **pure**. In other scenarios, do not change the default value **partial** of this parameter.

Default value: partial

enable_codegen_print

Parameter description: Specifies whether the LLVM IR function can be printed in logs.

Type: USERSET

Value range: Boolean

- **on** indicates that the LLVM IR function can be printed in logs.
- **off** indicates that the LLVM IR function cannot be printed in logs.

Default value: off

codegen_cost_threshold

Parameter description: The LLVM compilation takes some time to generate executable machine code. Therefore, LLVM compilation is beneficial only when the actual execution cost is more than the sum of the code required for generating machine code and the optimized execution cost. This parameter specifies a threshold. If the estimated execution cost exceeds the threshold, LLVM optimization is performed.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 10000

enable_constraint_optimization

Parameter description: Specifies whether the informational constraint optimization execution plan can be used for an HDFS foreign table.

Type: SUSET

Value range: Boolean

- **on** indicates the plan can be used.
- **off** indicates the plan cannot be used.

Default value: on

enable_bloom_filter

Parameter description: Specifies whether the BloomFilter optimization is used.

Type: USERSET

Value range: Boolean

- **on** indicates the BloomFilter optimization can be used.
- **off** indicates 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.

Type: SUSET

Value range: Boolean

- **on** indicates that the extrapolation logic is used for data of DATE type based on historical statistics.
- **off** indicates that the extrapolation logic is not used for data of DATE type based on historical statistics.

Default value: off

autoanalyze

Parameter description: Specifies whether to automatically collect statistics on tables that have no statistics when a plan is generated. **autoanalyze** cannot be used for foreign or temporary tables. To collect statistics, manually perform the ANALYZE operation. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. In this case, manually perform the ANALYZE operation on the table to synchronize statistics.

Type: SUSET

Value range: Boolean

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

Default value: off

query_dop

Parameter description: Specifies the user-defined degree of parallelism.

Type: USERSET

Value range: an integer ranging from -64 to 64.

[1, 64]: Fixed SMP is enabled, and the system will use the specified degree.

0: SMP adaptation function is enabled. The system dynamically selects the optimal parallelism degree [1,8] (x86 platforms) or [1,64] (Huawei Kunpeng platforms) for each query based on the resource usage and query plans.

[-64, -1]: SMP adaptation is enabled, and the system will dynamically select a degree from the limited range.

 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 shown as 0 in **pg_class** during plan generation.

Type: SUSET

Value range: Boolean

- **on** enables the check.
- **off** disables the check.

Default value: on

enable_sonic_hashagg

Parameter description: Specifies whether to use the Hash Agg operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Agg operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Agg operator is not used for column-oriented hash table design.

 NOTE

- If **enable_sonic_hashagg** is enabled and certain constraints are met, the Hash Agg operator will be used for column-oriented hash table design, and the memory usage of the operator can be reduced. However, in scenarios where the code generation technology (enabled by [enable_codegen](#)) can significantly improve performance, the performance of the operator may deteriorate.
- If **enable_sonic_hashagg** is enabled and certain constraints are met, the Hash Agg operator will be used for column-oriented hash table design, and the **Sonic Hash Aggregation** operator will be displayed in the output of the Explain Analyze/Performance operation. If the constraints are not met, the **Hash Aggregation** operator will be displayed. For details, see [Querying SQL Statements That Affect Performance Most](#).

Default value: on

enable_sonic_hashjoin

Parameter description: Specifies whether to use the Hash Join operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Join operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Join operator is not used for column-oriented hash table design.

 NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable_sonic_hashjoin** is enabled, the memory usage of 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 certain constraints are met, the Hash Join operator will be used for column-oriented hash table design, and the **Sonic Hash Join** operator will be displayed in the output of the Explain Analyze/Performance operation. If the constraints are not met, the **Hash Join** operator will be displayed. For details, see [Querying SQL Statements That Affect Performance Most](#).

Default value: on

plan_cache_mode

Parameter description: Specifies the policy for generating an execution plan in the **prepare** statement.

Type: USERSET

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.
- **force_custom_plan** indicates that the **custom plan** is forcibly used.

 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.
- **custom plan** is a plan generated after you run a **prepare** statement where parameters in the execute statement is embedded in the **prepare** statement. 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.
- **generic plan** is a plan generated for 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 solution 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 parameter field. When some bound parameters are used, the plan execution performance is poor.

Default value: auto

13.10 Error Reporting and Logging

13.10.1 Logging Time

client_min_messages

Parameter description: Specifies which level of messages are sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

Type: USERSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Valid values: Enumerated values. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error** For details about the parameters, see [Table 13-3](#).

Default value: notice

log_min_messages

Parameter description: Specifies which level of messages are written to the server log. Each level covers all the levels following it. The later the level, the fewer messages are sent to the log.

Type: SUSERSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Value range: enumerated type. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 13-3](#).

Default value: warning

log_min_error_statement

Parameter description: Specifies which SQL statements that cause errors condition will be recorded in the server log.

Type: SUSERSET

Value range: enumerated type. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 13-3](#).

 NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic**: This feature is disabled.

Default value: error

log_min_duration_statement

Parameter description: Specifies the threshold for logging statement execution durations. The execution duration that is greater than the specified value will be logged.

This parameter helps track query statements that need to be optimized. For clients using extended query protocol, durations of the Parse, Bind, and Execute are logged independently.

Type: SUSET

NOTICE

If this parameter and **log_statement** are used at the same time, statements recorded based on the value of **log_statement** will not be logged again after their execution duration exceeds the value of this parameter. 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 current statement message to the last logged duration.

Value range: an integer ranging from -1 to INT_MAX. The unit is millisecond.

- If this parameter is set to **250**, execution durations of SQL statements that run 250 ms or longer will be logged.
- **0**: Execution durations of all the statements are logged.
- **-1**: This feature is disabled.

Default value: 30 min

backtrace_min_messages

Parameter description: Prints the function's stack information to the server's log file if the level of information generated is greater than or equal to this parameter level.

Type: SUSET

NOTICE

This parameter is used for locating customer on-site problems. Because frequent stack printing will affect the system's overhead and stability, therefore, when you locate the onsite problems, set the value of this parameter to ranks other than **fatal** and **panic**.

Value range: enumerated values

Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 13-3](#).

Default value: panic

[Table 13-3](#) explains message severity used by GaussDB(DWS). If logging output is sent to **syslog** or **eventlog**, severity is translated in GaussDB(DWS) as shown in the table.

Table 13-3 Message Severity Levels

Severity	Description	syslog	eventlog
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATIO N
log	Reports information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATIO N
info	Provides information implicitly requested by the user, for example, output from VACUUM VERBOSE .	INFO	INFORMATIO N
notice	Provides information that might be helpful to users, for example, notice of truncation of long identifiers and index created as part of the primary key.	NOTICE	INFORMATIO N
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

plog_merge_age

Parameter description: Specifies the output interval of performance log data.

Type: SUSET

NOTICE

This parameter value is in milliseconds. You are advised to set this parameter to a value that is a multiple of 1000. That is, the value is in seconds. Name extension of the performance log files controlled by this parameter is .prf. These log files are stored in the **\$GAUSSLOG/gs_profile/<node_name>** directory. *node_name* is the value of **pgxc_node_name** in the **postgres.conf** file. You are advised not to use this parameter externally.

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

- **0** indicates that the current session will not output performance log data.
- A value other than 0 indicates the output interval of performance log data. The smaller the value is, the more log data is output, resulting in more negative impact on the performance.

Default value: 3s

13.10.2 Logging Content

debug_pretty_print

Parameter description: Specifies the logs produced by **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan**. The output format is more readable but much longer than the output generated when this parameter is set to **off**.

Type: USERSET

Value range: Boolean

- **on** indicates the indentation is enabled.
- **off** indicates the indentation is disabled.

Default value: on

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.

Type: SUSET

Value range: Boolean

- If this parameter is set to **off**, the difference between setting this parameter and setting **log_min_duration_statement** is that exceeding **log_min_duration_statement** forces the text of the query to be logged, but this parameter does not.
- If this parameter is set to **on** and **log_min_duration_statement** has a positive value, all durations are 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: on

log_error_verbosity

Parameter description: Specifies the amount of detail written in the server log for each message that is logged.

Type: SUSET

Value range: enumerated values

- **terse** indicates that the output excludes the logging of 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 logging of 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_lock_waits

Parameter description: If the time that a session used to wait 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.

Type: SUSET

Value range: Boolean

- **on** indicates the information is recorded.
- **off** indicates the information is not recorded.

Default value: off

log_statement

Parameter description: Specifies whether to record SQL statements. 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).

Type: SUSET

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 the extended query protocol is used, this setting also does not log statements before the execution phase (during parse analysis or planning). 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 are recorded. The PREPARE, EXECUTE, and EXPLAIN ANALYZE statements are also recorded.

Default value: none

log_temp_files

Parameter description: Specifies whether to record the delete information of temporary files. Temporary files can be created for sorting, hashing, and temporary querying results. A log entry is generated for each temporary file when it is deleted.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB.

- A positive value indicates that the delete information of temporary files whose values are larger than that of **log_temp_files** is recorded.
- If the parameter is set to **0**, all the delete information of temporary files is recorded.
- If the parameter is set to **-1**, the delete information of no temporary files is recorded.

Default value: -1

logging_module

Parameter description: Specifies whether module logs can be output on the server. This parameter is a session-level parameter, and you are not advised to use the **gs_guc** tool to set it.

Type: USERSET

Value range: a string

Default value: off. All the module logs on the server can be viewed by running **show logging_module**.

Setting method: First, you can run **show logging_module** to view which module is controllable. For example, the query output result is as follows:

```
show logging_module;
logging_module
-----
ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM)
(1 row)
```

Controllable modules are identified by uppercase letters, and the special ID ALL is used for setting all module logs. You can control module logs to be exported by setting the log modules to **on** or **off**. Enable log output for SSL:

```
set logging_module='on(SSL)';
SET
show
logging_module;
logging_module
-----
ALL,on(SSL),off(DFS,GUC,HDFS,ORC,CARBONDATA,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ)
(1 row)
```

SSL log output is enabled.

The ALL identifier is equivalent to a shortcut operation. That is, logs of all modules can be enabled or disabled.

```
set logging_module='off(ALL)';
SET
show
logging_module;
logging_module
-----
ALL,on(),off(DFS,GUC,HDFS,ORC,CARBONDATA,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ)
(1 row)

set logging_module='on(ALL)';
SET
show
logging_module;
logging_module
-----
ALL,on(DFS,GUC,HDFS,ORC,CARBONDATA,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ),off()
(1 row)
```

Dependency relationship: The value of this parameter depends on the settings of [log_min_messages](#).

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.

Type: SUSET

Value range: Boolean

- **on:** Statements not pushed down will be logged.
- **off:** Statements not pushed down will not be logged.

Default value: on

13.11 Statistics During the Database Running

13.11.1 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 an 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 families 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.

Type: SUSET

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 data activities.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

NOTE

When the database to be cleaned up is selected from the AutoVacuum automatic cleanup process, the database statistics are required. In this case, the default value is set to **on**.

Default value: on

track_io_timing

Parameter description: Collects statistics about I/O invoking timing in the database. The I/O timing statistics can be queried by using the **pg_stat_database** parameter.

Type: SUSET

Value range: Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the OS at the current time. As a result, large numbers 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 about invoking times and duration in a function.

Type: SUSET

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 and C language functions are traced.
- **none** indicates that the function tracing function is disabled.

Default value: none

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.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: off

track_thread_wait_status_interval

Parameter description: Specifies the interval of collecting the thread status information periodically.

Type: SUSET

Value range: an integer ranging from 0 to 1440. The unit is minute (min).

Default value: 30min

enable_save_datachanged_timestamp

Parameter description: Specifies whether to record the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** is performed on table data.

Type: USERSET

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: Specifies whether to collect statistics on the number of the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** statements that are being executed in each session, the response time of the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements, and the number of DDL, DML, and DCL statements.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: off

 NOTE

- The **track_sql_count** parameter is restricted by the **track_activities** parameter when the **gs_sql_count** or **pgxc_sql_count** view is queried.
 - If **track_activities** is set to **on** and **track_sql_count** is set to **off**, a warning message indicating that **track_sql_count** is disabled will be displayed when the view **gs_sql_count**, **pgxc_sql_count**, **gs_workload_sql_count**, **pgxc_workload_sql_count**, **global_workload_sql_count**, **gs_workload_sql_elapse_time**, **pgxc_workload_sql_elapse_time**, or **global_workload_sql_elapse_time** are queried.
 - If both **track_activities** and **track_sql_count** are set to **off**, two logs indicating that **track_activities** is disabled and **track_sql_count** is disabled will be displayed when the views are queried.
 - If **track_activities** is set to **off** and **track_sql_count** is set to **on**, a log indicating that **track_activities** is disabled will be displayed when the views are queried.
- If **track_sql_count** is set to **off**, querying the **gs_sql_count** or **pgxc_sql_count** view returns 0.

13.11.2 Performance Statistics

During the running of the database, the lock access, disk I/O operation, and invalid message process are involved. All these operations are the bottleneck of the database performance. The performance statistics method provided by GaussDB(DWS) can facilitate the performance fault location.

Generating Performance Statistics Logs

Parameter description: For each query, the following four parameters control the performance statistics of corresponding modules recorded in the server log:

- The **og_parser_stats** parameter controls the performance statistics of a parser recorded in the server log.
- The **log_planner_stats** parameter controls the performance statistics of a query optimizer recorded in the server log.
- The **log_executor_stats** parameter controls the performance statistics of an executor recorded in the server log.
- The **log_statement_stats** parameter controls the performance statistics of the whole statement recorded in the server log.

All these parameters can only provide assistant analysis for administrators, which are similar to the `getrusage()` of the Linux OS.

Type: SUSET

NOTICE

- **log_statement_stats** records the total statement statistics while other parameters only record statistics about each statement.
- The **log_statement_stats** parameter cannot be enabled together with other parameters recording statistics about each statement.

Value range: Boolean

- **on** indicates the function of recording performance statistics is enabled.
- **off** indicates the function of recording performance statistics is disabled.

Default value: off

13.12 Workload Management

If database resource usage is not controlled, concurrent tasks easily preempt resources. As a result, the OS will be overloaded and cannot respond to user tasks; or even crash and cannot provide any services to users. The GaussDB(DWS) workload management function balances the database workload based on available resources to avoid database overloading.

cgroup_name

Parameter description: Specifies the name of the Cgroup in use. It can be used to change the priorities of jobs in the queue of a Cgroup.

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups associated with **cgroup_name** take effect.

If the Workload Cgroup level is specified during the **cgroup_name** change, the database does not check the Cgroup level. The level ranges from 1 to 10.

Type: USERSET

You are not advised to set **cgroup_name** and **session_respool** at the same time.

Value range: a string

Default value: DefaultClass:Medium

NOTE

DefaultClass:Medium indicates the **Medium** Cgroup belonging to the **Timeshare** Cgroup under the **DefaultClass** Cgroup.

enable_cgroup_switch

Parameter description: Specifies whether the database automatically switches to the **TopWD** group when executing statements by group type.

Type: USERSET

Value range: Boolean

- **on:** The database automatically switches to the **TopWD** group when executing statements by group type.
- **off:** The database does not automatically switch to the **TopWD** group when executing statements by group type.

Default value: off

memory_tracking_mode

Parameter description: Specifies the memory information recording mode.

Type: USERSET

Valid value:

- **none:** Memory statistics is not collected.
- **normal:** Only memory statistics is collected in real time and no file is generated.
- **executor:** The statistics file is generated, containing the context information about all allocated memory used by the execution layer.
- **fullexec:** The generated file includes the information about all memory contexts requested by the execution layer.

Default value: none

memory_detail_tracking

Parameter description: Specifies the sequence number of the memory background information distributed in the needed thread and **plannodeid** of the query where the current thread is located.

Type: USERSET

Value range: a string

Default value: empty

NOTICE

It is recommended that you retain the default value for this parameter.

enable_instance_metric_persistent

Parameter description: Specifies whether the instance resource monitoring dumping function is enabled. When this function is enabled, the instance monitoring data is saved to the system catalog [GS_WLM_INSTANCE_HISTORY](#).

Type: SIGHUP

Value range: Boolean

- **on** indicates that the instance resource monitoring dumping function is enabled.
- **off:** Specifies that the instance resource monitoring dumping function is disabled.

Default value: on

resource_track_level

Parameter description: Specifies the resource monitoring level of the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

Type: USERSET

Value range: enumerated values

- **none**: Resources are not monitored.
- **query**: Resources used at the query level are monitored.
- **operator**: Resources used at query and operator levels are monitored.

Default value: query

resource_track_cost

Parameter description: Specifies the minimum execution cost for resource monitoring on statements in the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

- -1 indicates that resource monitoring is disabled.
- A value ranging from 0 to 9 indicates that statements whose execution cost is greater than or equal to 10 will be monitored.
- A value greater than or equal to 10 indicates that statements whose execution cost exceeds this value will be monitored.

Default value: 100000

resource_track_duration

Parameter description: Specifies the minimum statement execution time that determines whether information about jobs of a statement recorded in the real-time view (see [Table 14-4](#)) will be dumped to a historical view after the statement is executed. Job information will be dumped from the real-time view (with the suffix **statistics**) to a historical view (with the suffix **history**) if the statement execution time is no less than this value.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is second.

- 0 indicates that information about all statements recorded in the real-time resource monitoring view (see [Table 14-4](#)) will be archived into historical views.
- If the value is greater than 0, information about statements recorded in the real-time resource monitoring view (see [Table 14-4](#)), whose execution time exceeds this value will be archived into historical views.

Default value: 1 min

disable_memory_protect

Parameter description: Stops memory protection. To query system views when system memory is insufficient, set this parameter to **on** to stop memory protection. This parameter is used only to diagnose and debug the system when system memory is insufficient. Set it to **off** in other scenarios.

Type: USERSET

Value range: Boolean

- **on** indicates that memory protection stops.
- **off** indicates that memory is protected.

Default value: off

query_band

Parameter description: Specifies the job type of the current session.

Type: USERSET

Value range: a string

Default value: empty

bbox_dump_count

Parameter description: Specifies the maximum number of core files that are generated by GaussDB(DWS) and can be stored in the path specified by **bbox_dump_path**. If the number of core files exceeds this value, old core files will be deleted. This parameter is valid only if **enable_bbox_dump** is set to **on**.

Type: USERSET

Value range: an integer ranging from 1 to 20

Default value: 8

NOTE

When core files are generated during concurrent SQL statement execution, the number of files may be larger than the value of **bbox_dump_count**.

io_limits

Parameter description: Specifies the upper limit of IOPS triggered.

Type: USERSET

Value range: an integer ranging from 0 to 1073741823

Default value: 0

io_priority

Parameter description: Specifies the I/O priority for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.

Type: USERSET

Value range: enumerated values

- **None** indicates no control.
- **Low** indicates that the IOPS is reduced to 20% of the original value.
- **Medium** indicates that the IOPS is reduced to 50% of the original value.
- **High** indicates that the IOPS is reduced to 80% of the original value.

Default value: None

session_respool

Parameter description: Specifies the resource pool associated with the current session.

Type: USERSET

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups associated with **cgroup_name** take effect.

If the Workload Cgroup level is specified during the **cgroup_name** change, the database does not check the Cgroup level. The level ranges from 1 to 10.

You are not advised to set **cgroup_name** and **session_respool** at the same time.

Value range: a string. This parameter can be set to the resource pool configured through **create resource pool**.

Default value: invalid_pool

enable_transaction_parctl

Parameter description: whether to control transaction block statements and stored procedure statements.

Type: USERSET

Value range: Boolean

- **on:** Transaction block statements and stored procedure statements are controlled.
- **off:** Transaction block statements and stored procedure statements are not controlled.

Default value: on

transaction_pending_time

Parameter description: maximum queuing time of transaction block statements and stored procedure statements if **enable_transaction_parctl** is set to **on**.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is second.

- **-1 or 0:** No queuing timeout is specified for transaction block statements and stored procedure statements. The statements can be executed when resources are available.
- **Value greater than 0:** If transaction block statements and stored procedure statements have been queued for a time longer than the specified value, they are forcibly executed regardless of the current resource situation.

Default value: 0

NOTICE

This parameter is valid only for internal statements of stored procedures and transaction blocks. That is, this parameter takes effect only for the statements whose **enqueue** value (for details, see [PG_SESSION_WLMSTAT](#)) is **Transaction** or **StoredProc**.

13.13 Default Settings of Client Connection

13.13.1 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 (,).

Type: USERSET

- If the schema of a temporary table exists in the current session, the scheme can be listed in **search_path** by using the alias **pg_temp**, for example, '**pg_temp,public**'. The schema of a temporary table has the highest search priority and is always searched before all the 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 only searched for database objects, including tables, views, and data types. Functions or operator names will not be searched for.
- The schema of a system catalog, **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 specifying a particular schema, the object will be placed in the first valid schema listed in **search_path**. An error will be 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**, a database can be shared (where no users have private schemas, and all share use of public), and 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.

Type: USERSET

Value range: a string

Default value: "**\$user**",**public**

 NOTE

\$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **\$user** will be ignored.

default_tablespace

Parameter description: Specifies the default tablespace of the created objects (tables and indexes) when a **CREATE** command does not explicitly specify a tablespace.

- The value of this parameter is either the name of a tablespace, or an empty string that specifies 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, the [temp_tablespaces](#) is consulted instead.
- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

Type: USERSET

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.

- **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.

Type: USERSET

Value range: a string

Default value: installation

temp_tablespaces

Parameter description: Specifies tablespaces to which temporary objects will be created (temporary tables and their indexes) when a **CREATE** command does not explicitly specify a tablespace. Temporary files for sorting large data are created in these tablespaces.

The value of this parameter is a list of names of tablespaces. When there is more than one name in the list, GaussDB(DWS) chooses a random tablespace from the list upon the creation of a temporary object each time. Except that 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(DWS) will automatically use the default tablespace of the current database instead.

Type: USERSET

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.

Type: USERSET

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.

Type: USERSET

Value range: enumerated values

- **READ COMMITTED:** Only committed data is read. This is the default.
- **READ UNCOMMITTED:** GaussDB(DWS) does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.
- **REPEATABLE READ:** Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE:** GaussDB(DWS) does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: READ COMMITTED

default_transaction_deferrable

Parameter description: Specifies the default delaying state of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

GaussDB(DWS) does not support the serializable isolation level of each transaction. The parameter is insignificant.

Type: USERSET

Value range: Boolean

- **on** indicates a transaction is delayed by default.
- **off** indicates 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.

Type: USERSET

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 when the server receives the command) is longer than the duration specified by the parameter, error information is displayed when you attempt to execute the statement and the statement then exits.

Type: USERSET

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: 0

vacuum_freeze_min_age

Parameter description: Specifies the minimum cutoff age (in the same transaction), based on which **VACUUM** decides whether to replace transaction IDs with FrozenXID while scanning a table.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

NOTE

Although you can set this parameter to a value ranging from 0 to 1000000000 anytime, **VACUUM** will limit the effective value to half the value of `autovacuum_freeze_max_age` by default.

Default value: 5000000000

vacuum_freeze_table_age

Parameter description: Specifies the time that **VACUUM** freezes tuples while scanning the whole table. **VACUUM** performs a whole-table scan if the value of the `pg_class.relfrozenxid` column of the table has reached the specified time.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

NOTE

Although users can set this parameter to a value ranging from 0 to 2000000000 anytime, **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: 15000000000

bytea_output

Parameter description: Specifies the output format for values of the bytea type.

Type: USERSET

Value range: enumerated values

- **hex** indicates the binary data is converted to the two-byte hexadecimal digit.
- **escape** indicates 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.

Type: USERSET

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.

Type: USERSET

Value range: enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

Default value: content

gin_pending_list_limit

Parameter description: Specifies the maximum size of the GIN pending list which is used when **fastupdate** is enabled. If the list grows larger than this maximum size, it is cleaned up by moving the entries in it to the main GIN data structure in batches. This setting can be overridden for individual GIN indexes by modifying index storage parameters.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: 4 MB

13.13.2 Zone 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.

Type: USERSET

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](#).

Suggestion: 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 others. This may lead to incorrect time zone conversion and cause errors.

IntervalStyle

Parameter description: Specifies the display format for interval values.

Type: USERSET

Value range: enumerated values

- **sql_standard** indicates that output matching SQL standards will be generated.
- **postgres** indicates that output matching PostgreSQL 8.4 will be generated when the [DateStyle](#) parameter is set to **ISO**.
- **postgres_verbose** indicates that output matching PostgreSQL 8.4 will be generated when the [DateStyle](#) parameter is set to **non_ISO**.
- **iso_8601** indicates that output matching the time interval "format with designators" defined in ISO 8601 will be generated.
- **oracle** indicates the output result that matches the numtodsinterval function in the Oracle database. For details, see numtodsinterval.

NOTICE

The **IntervalStyle** parameter also affects the interpretation of ambiguous interval input.

Default value: postgres

timezone_abbreviations

Parameter description: Specifies the time zone abbreviations that will be accepted by the server.

Type: USERSET

Value range: a string. You can obtain it by querying the pg_timezone_names view.

Default value: Default

NOTE

Default indicates an abbreviation that works 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: Specifies 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).

Type: USERSET

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 type (character set).

Set this parameter as needed. Try to keep the client code and server code consistent to improve efficiency.

Type: USERSET

Value range: encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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 a 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.

Valid values depend on the current system. On some systems, this zone 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.

Type: SUSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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`. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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_numeric

Parameter description: Specifies the display format of numbers. It affects the output of functions such as `to_char`. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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_time

Parameter description: Specifies the display format of time and zones. It affects the output of functions such as `to_char`. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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

default_text_search_config

Parameter description: Specifies the text search configuration.

If the specified text search configuration does not exist, an error will be reported. If the specified text search configuration is deleted, set **default_text_search_config** again. Otherwise, an error will be reported, indicating incorrect configuration.

- The text search configuration is used by text search functions that do not have an explicit argument specifying the configuration.
- When a configuration file matching the environment is determined, **gs_initdb** will initialize the configuration file with a setting that corresponds to the environment.

Type: USERSET

Value range: a string

NOTE

GaussDB(DWS) supports the following two configurations: `pg_catalog.english` and `pg_catalog.simple`.

Default value: `pg_catalog.english`

13.13.3 Other Default Parameters

This section describes the default database loading parameters of the database system.

dynamic_library_path

Parameter description: Specifies the path for saving the shared database files that are dynamically loaded for data searching. When a dynamically loaded

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.

The value of **dynamic_library_path** must be a list of absolute paths separated by colons (:) or by semi-colons (;) on the Windows OS. The special variable **\$libdir** in the beginning of a path will be replaced with the module installation directory provided by GaussDB(DWS). For example:

```
dynamic_library_path = '/usr/local/lib/postgresql/opt/testgs/lib:$libdir'
```

Type: SUSET

Value range: a string

NOTE

If the value of this parameter is set to an empty character string, the automatic path search is turned off.

Default value: **\$libdir**

gin_fuzzy_search_limit

Parameter description: Specifies the upper limit of the size of the set returned by GIN indexes.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 0

13.14 Lock Management

In GaussDB(DWS), a deadlock may occur when concurrently executed transactions compete for resources. This section describes parameters used for managing transaction lock mechanisms.

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.

- 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 released before the waiter decides to check for deadlocks will be reduced.
- When **log_lock_waits** is set, this parameter also determines the duration you need to wait before a log message about the lock wait is issued. If you are

trying to investigate locking delays, you need to set this parameter to a value smaller than normal **deadlock_timeout**.

Type: SUSET

Value range: an integer ranging from 1 to INT_MAX. The unit is millisecond (ms).

Default value: 1s

lockwait_timeout

Parameter description: Specifies the longest time to wait before a single lock times out. If the time you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: 20 min

update_lockwait_timeout

Parameter description: sets 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 you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: 2min

partition_lock_upgrade_timeout

Parameter description: Specifies the time to wait before the attempt of a lock upgrade from ExclusiveLock to AccessExclusiveLock times out on partitions.

- 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, ExclusiveLock is acquired for the partitions during data rearrangement and AccessExclusiveLock is acquired during file exchange.
- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period of time longer than specified by the [lockwait_timeout](#) parameter.
- When doing MERGE PARTITION or CLUSTER PARTITION on a partitioned table, you need to acquire AccessExclusiveLock during file exchange. If the lock fails to be acquired, the acquisition is retried in 50 ms. This 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.

Type: USERSET

Value range: an integer ranging from -1 to 3000. The unit is second (s).

Default value: 1800

13.15 Version and Platform Compatibility

13.15.1 Compatibility with Earlier Versions

This section describes the parameter control of the downward compatibility and external compatibility features of GaussDB(DWS). Backward compatibility of the database system provides support for the application of databases of earlier versions. This section describes parameters used for controlling backward compatibility of a database.

array_nulls

Parameter description: Determines whether the array input parser recognizes unquoted NULL as a null array element.

Type: USERSET

Value range: Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing **NULL** values can still be created when this parameter is set to **off**.

Default value: on

backslash_quote

Parameter description: Determines whether a single quotation mark can be represented by \ in a string text.

Type: USERSET

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: Determines 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.

Type: USERSET

Value range: Boolean

- **on** indicates **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates **CREATE TABLE** and **CREATE TABLE AS** cannot include any **OID** field in newly-created tables.

Default value: off

escape_string_warning

Parameter description: Specifies a warning on directly using a backslash (\) 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 is now to treat the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate codes that need to be changed.

Type: USERSET

Value range: Boolean

Default value: on

lo_compat_privileges

Parameter description: Determines whether to enable backward compatibility for the privilege check of large objects.

Type: SUSET

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.

Default value: off

quote_all_identifiers

Parameter description: When the database generates SQL, this parameter forcibly quotes all identifiers even if they are not keywords. This will affect the output of EXPLAIN as well as the results of functions, such as pg_get_viewdef. For details, see the **--quote-all-identifiers** parameter of **gs_dump**.

Type: USERSET

Value range: Boolean

- **on** indicates the forcible quotation function is enabled.
- **off** indicates the forcible quotation function is disabled.

Default value: off

sql_inheritance

Parameter description: Determines whether to inherit semantics.

Type: USERSET

Value range: Boolean

off indicates that child tables cannot be accessed by various commands. That is, an ONLY keyword is used by default. This setting is compatible with versions earlier than PostgreSQL 7.1.

Default value: on

standard_conforming_strings

Parameter description: Determines 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'...').

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: on

synchronize_seqscans

Parameter description: Controls sequential scans of tables to synchronize with each other. Concurrent scans read the same data block about at the same time and share the I/O workload.

Type: USERSET

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: Controls whether certain limited features, such as GDS table join, are available. These features are not explicitly prohibited in earlier versions, but are not recommended due to their limitations in certain scenarios.

Type: USERSET

Value range: Boolean

- **on** indicates that the features are enabled and forward compatible, but may incur errors in certain scenarios.
- **off** indicates that the features are disabled.

Default value: off

13.15.2 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: Determines whether expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`. They return true if `expr` evaluates to **NULL**, 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.

Type: USERSET

Value range: Boolean

- **on** indicates expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`.
- **off** indicates `expr = NULL` always returns **NULL**.

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.

td_compatible_truncation

Parameter description: Determines whether to enable features compatible with a Teradata database. You can set this parameter to **on** when connecting to a database compatible with the Teradata database, so that when you perform the INSERT operation, overlong strings are truncated based on the allowed maximum length before being inserted into char- and varchar-type columns in the target table. This ensures all data is inserted into the target table without errors reported.

 NOTE

- The string truncation function cannot be used if the **INSERT** statement includes a foreign table.
- If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (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.

Type: USERSET

Value range: Boolean

- **on** indicates overlong strings are truncated.
- **off** indicates overlong strings are not truncated.

Default value: off

13.16 Fault Tolerance

This section describes parameters used for controlling the methods that the server processes an error occurring in the database system.

exit_on_error

Parameter description: Specifies whether to terminate the current session.

Type: SUSERSET

Value range: Boolean

- **on** indicates that any error will terminate the current session.
- **off** indicates that only a FATAL error will terminate the current session.

Default value: off

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, the occurring character encoding conversion errors will be recorded in logs. Additionally, converted characters that have conversion errors will be ignored and replaced with question marks (?).

Type: USERSET

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

max_query_retry_times

Parameter description: Specifies the maximum number of automatic retries for an SQL statement having execution errors. Currently, supported error types include "Connection reset by peer", "Lock wait timeout", and "Connection timed out". If it is set to **0**, the retry function is disabled.

Type: USERSET

Value range: an integer ranging from 0 to 20

Default value: 6

retry_ecode_list

Parameter description: Specifies the list of SQL error types that support automatic retry.

Type: USERSET

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 CG003 CG004

13.17 Cluster Transaction Parameters

This section describes the settings and value ranges of cluster transaction parameters.

transaction_isolation

Parameter description: Specifies the isolation level of the current transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 13-1](#).

Value range:

- **READ COMMITTED:** Only committed data is read. This is the default.
- **READ UNCOMMITTED:** GaussDB(DWS) does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.
- **REPEATABLE READ:** Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE:** GaussDB(DWS) does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: READ COMMITTED

transaction_read_only

Parameter description: Specifies that the current transaction is a read-only transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 13-1](#).

Value range: Boolean

- **on** indicates that the current transaction is a read-only transaction.
- **off** indicates that the current transaction can be a read/write transaction.

Default value: off

xc_maintenance_mode

Parameter description: Specifies whether the system is in maintenance mode.

This parameter is a SUSET parameter. Set it based on method 3 in [Table 13-1](#).

Value range: Boolean

- **on** indicates that maintenance mode is enabled.
- **off** indicates that the maintenance mode is disabled.

NOTICE

Enable the maintenance mode with caution to avoid cluster data inconsistencies.

Default value: off

allow_concurrent_tuple_update

Parameter description: Specifies whether to allow concurrent update.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 13-1](#).

Value range: Boolean

- **on** indicates that concurrent update is allowed.
- **off** indicates that concurrent update is disallowed.

Default value: on

gtm_backup_barrier

Parameter description: Specifies whether to create a restoration point for the GTM starting point.

This parameter is a SUSET parameter. Set it based on instructions provided in [Table 13-1](#).

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

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. Currently, this parameter is not used in GaussDB(DWS). Similar to this parameter, the [default_transaction_deferrable](#) parameter is used to specify whether to allow delayed execution of a transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 13-1](#).

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

enforce_two_phase_commit

Parameter description: This parameter is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

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/true**, all versions of tuples in the table are displayed.

This parameter is a USERSET parameter. Set it based on instructions provided in [Table 13-1](#).

Value range: Boolean

- **on/true** indicates that all versions of tuples in the table are displayed.
- **off/false** indicates that no versions of tuples in the table are displayed.

Default value: off

enable_redistribute

Parameter description: Specifies whether unmatched nodes are redistributed.

This parameter is a SUSERSET parameter. Set it based on instructions provided in [Table 13-1](#).

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 what nodes are deployed for the HA mode. The nodes can consist of a primary, a standby, and a secondary node; or consist of a primary and multiple standby nodes.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in [Table 13-1](#).

This parameter is an internal parameter used for CM deployment. Do not set it.

Value range: an integer

- **1** indicates that the HA nodes consist of a primary and multiple standby nodes.
- **0**: indicates that the HA nodes consist of a primary, a standby, and a secondary node.

Default value: 0

enable_gtm_free

Parameter description: Specifies whether the GTM-FREE mode is enabled. In large concurrency scenarios, the snapshots delivered by the GTM increase in number and size. The network between the GTM and the CN becomes the performance bottleneck. The GTM-FREE mode is used to eliminate the bottleneck. In this mode, the CN communicates with DNs instead of the GTM. The CN sends queries to each DN, which locally generates snapshots and xids, ensuring external write consistency but not external read consistency.

You are not advised to set this parameter to **on** in OLTP or OLAP scenarios where strong read consistency is required. This parameter is invalid for GaussDB(DWS).

This parameter is a SUSE parameter. Set it based on instructions provided in [Table 13-1](#).

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

13.18 Developer Operations

enable_fast_query_shipping

Parameter description: Specifies whether to use the distributed framework for a query planner.

Type: USERSET

Value range: Boolean

- **on** indicates that execution plans are generated on CNs and DNs separately.
- **off** indicates that the distributed framework is used. Execution plans are generated on CNs and then sent to DN for execution.

Default value: on

enable_trigger_shipping

Parameter description: Specifies whether the trigger can be pushed to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the trigger can be pushed to DN for execution.
- **off** indicates that the trigger cannot be pushed to DN. It must be executed on the CN.

Default value: on

enable_remotejoin

Parameter description: Specifies whether JOIN operation plans can be delivered to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that JOIN operation plans can be delivered to DN for execution.
- **off** indicates that JOIN operation plans cannot be delivered to DN for execution.

Default value: on

enable_remotegroup

Parameter description: Specifies whether the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DN for execution.
- **off** indicates that the execution plans of **GROUP BY** and **AGGREGATE** cannot be delivered to DN for execution.

Default value: on

enable_remotelimit

Parameter description: Specifies whether the execution plan specified in the LIMIT clause can be pushed down to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan specified in the LIMIT clause can be pushed down to DN for execution.
- **off** indicates that the execution plan specified in the LIMIT clause cannot be delivered to DN for execution.

Default value: on

enable_remotesort

Parameter description: Specifies whether the execution plan of the ORDER BY clause can be delivered to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan of the ORDER BY clause can be delivered to DN for execution.
- **off** indicates that the execution plan of the ORDER BY clause cannot be delivered to DN for execution.

Default value: on

enable_join_pseudoconst

Parameter description: Specifies whether joining with the pseudo constant is allowed. A pseudo constant indicates that the variables on both sides of a join are identical to the same constant.

Type: USERSET

Value range: Boolean

- **on** indicates that joining with the pseudo constant is allowed.
- **off** indicates that joining with the pseudo constant is not allowed.

Default value: off

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(DWS) compilation.

Type: USERSET

Value range: Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

 NOTE

This parameter is set to **on** by default if GaussDB(DWS) is compiled with various assertion checks enabled.

Default value: off

distribute_test_param

Parameter description: Specifies whether the embedded test stubs for testing the distribution framework take effect. In most cases, developers embed some test stubs in the code during fault injection tests. Each test stub is identified by a unique name. The value of this parameter is a triplet that includes three values: thread level, test stub name, and error level of the injected fault. The three values are separated by commas (,).

Type: USERSET

Value range: a string indicating the name of any embedded test stub.

Default value: -1, default, default

ignore_checksum_failure

Parameter description: Sets whether to ignore check failures (but still generates an alarm) and continues reading data. This parameter is valid only if **enable_crc_check** is set to **on**. Continuing reading data may result in breakdown, damaged data being transferred or hidden, failure of data recovery from remote nodes, or other serious problems. You are not advised to modify the settings.

Type: SUSERSET

Value range: Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

Default value: off

enable_colstore

Parameter description: Specifies whether to create a table as a column-store table by default when no storage method is specified. The value for each node must be the same. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSERSET

Value range: Boolean

Default value: off

enable_force_vector_engine

Parameter description: Specifies whether to forcibly generate vectorized execution plans for a vectorized execution operator if the operator's child node is a

non-vectorized operator. When this parameter is set to **on**, vectorized execution plans are forcibly generated.

Type: USERSET

Value range: Boolean

Default value: off

enable_csqual_pushdown

Parameter description: Specifies whether to deliver filter criteria for a rough check during query.

Type: SUSERSET

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: Specifies the name of a CSV file exported when [explain_perf_mode](#) is set to **run**.

Type: USERSET

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: NULL

explain_perf_mode

Parameter description: Specifies the display format of the **explain** command.

Type: USERSET

Value range: **normal**, **pretty**, **summary**, and **run**

- **normal** indicates that the default printing format is used.
- **pretty** indicates that the optimized display mode of GaussDB(DWS) is used. A new format contains a plan node ID, directly and effectively analyzing performance.
- **summary** indicates that the analysis result based on such 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.

Default value: pretty

trace_notify

Parameter description: Specifies whether to generate a large amount of debugging output for the **LISTEN** and **NOTIFY** commands. **client_min_messages** or **log_min_messages** must be **DEBUG1** or lower so that such output can be recorded in the logs on the client or server separately.

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: off

trace_sort

Parameter description: Specifies whether to display information about resource usage during sorting operations in logs. This parameter is available only when the macro **TRACE_SORT** is defined during the GaussDB(DWS) compilation. However, **TRACE_SORT** is currently defined by default.

Type: USERSET

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(DWS) to report an error, aborting the current transaction.

Type: SUSET

Value range: Boolean

Setting this parameter to **on** causes the system to report a warning, pad the damaged page with zeros, and then continue with subsequent 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** unless you do not want to restore data from the damaged pages of a table.

Default value: off

replication_test

Parameter description: Specifies whether to enable internal testing on the data replication function.

Type: USERSET

Value range: Boolean

- **on** indicates that internal testing on the data replication function is enabled.
- **off** indicates that internal testing on the data replication function is disabled.

Default value: off

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 (&) operations on the bit for each method. A method is selected if its value is not 0.

If **cost_param & 1** is not set to 0, an improvement mechanism is selected for calculating a non-equi join selection rate, which is more accurate in estimation of self-join (join between two same tables). In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized 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.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: 0

convert_string_to_digit

Parameter description: Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

Type: USERSET

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

Modify this parameter only when absolutely necessary because the modification will change the rule for converting internal data types and may cause unexpected results.

nls_timestamp_format

Parameter description: Specifies the default timestamp format.

Type: USERSET

Value range: a string

Default value: DD-Mon-YYYY HH:MI:SS.FF AM

enable_partitionwise

Parameter description: Specifies whether to select an intelligent algorithm for joining partitioned tables.

Type: USERSET

Value range: Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

Default value: off

max_user_defined_exception

Parameter description: Specifies the maximum number of exceptions. The default value cannot be changed.

Type: USERSET

Value range: an integer

Default value: 1000

datanode_strong_sync

Parameter description: This parameter no longer takes effect.

Type: USERSET

Value range: Boolean

- **on** indicates that forcible synchronization between stream nodes is enabled.
- **off** indicates that forcible synchronization between stream nodes is disabled.

Default value: off

enable_global_stats

Parameter description: Specifies the current statistics mode. This parameter is used to compare global statistics generation plans and the statistics generation

plans for a single DN. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates the global statistics mode.
- **off** or **false** indicates the single-DN statistics mode.

Default value: on

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.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for numeric data calculation is enabled.
- **off/false** indicates that optimization for numeric data calculation is disabled.

Default value: on

rewrite_rule

Parameter description: Specifies the rewriting rule for enabled optional queries. Some query rewriting rules are optional. Enabling them cannot always improve query efficiency. In a specific customer scenario, you can set the query rewriting rules through the 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. To set the parameters, you can perform the following operations:

```
set rewrite_rule=rule1;      --Enable query rewriting rule rule1.
set rewrite_rule=rule2,rule3; --Enable query rewriting rules rule2 and rule3.
set rewrite_rule=none;       --Disable all optional query rewriting rules.
```

Type: USERSET

Value range: a string

- **none:** Does not use any optional query rewriting rules.
- **lazyagg:** Uses the Lazy Agg query rewriting rules for eliminating aggregation operations in subqueries.
- **magicset:** Uses the Magic Set query rewriting rules (from the main query to subqueries).
- **uniquecheck:** Uses the Unique Check rewriting rule. (The scenario where the target column does not contain the expression sublink of the aggregate function can be improved. The function can be enabled only when the value of the target column is unique after the sublink is aggregated based on the

associated column. This function is recommended to be used by optimization engineers.)

- **disablerep**: Uses the function that prohibit pulling up sublinks of the replication table. (Disable sublink pull-up for the replication table.)

Default value: magicset

enable_compress_spill

Parameter description: Specifies whether to enable the compression function of writing data to a disk.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for writing data to a disk is enabled.
- **off/false** indicates that optimization for writing data to a disk is disabled.

Default value: on

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.

Type: USERSET

Value range: a string

- **LLVM_COMPILE** indicates that the codegen compilation time of each thread is displayed on the explain performance page.
- **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: off(ALL), which indicates that no location function is enabled.

resource_track_log

Parameter description: Specifies the log level of self-diagnosis. Currently, this parameter takes effect only in multi-column statistics.

Type: USERSET

Value range: a string

- **summary**: Brief diagnosis information is displayed.
- **detail**: Detailed diagnosis information is displayed.

Currently, the two parameter values differ only when there is an alarm about multi-column statistics not collected. If the parameter is set to **summary**, such an alarm will not be displayed. If it is set to **detail**, such an alarm will be displayed.

Default value: summary

pljava_vmoptions

Parameter description: Specifies the startup parameters for JVMs used by the PL/Java function.

Type: SUSET

Value range: a string, supporting:

- JDK8 JVM startup parameters.
- JDK8 JVM system attributes (starting with **-D**, for example, **-Djava.ext.dirs**).
- User-defined parameters (starting with **-D**, for example, **-Duser.defined.option**).

NOTICE

If **pljava_vmoptions** is set to a value beyond the value range, an error will be reported when PL/Java functions are used. For details, see [PL/Java User-defined Functions](#).

Default value: empty

enable_pbe_optimization

Parameter description: Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

Type: SUSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the query plan.
- **off** indicates that the optimization does not optimize the query plan.

Default value: on

enable_light_proxy

Parameter description: Specifies whether the optimizer optimizes the execution of simple queries on CNs.

Type: SUSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the execution.
- **off** indicates that the optimization does not optimize the execution.

Default value: on

enable_parallel_ddl

Parameter description: Controls whether multiple CNs can concurrently perform DDL operations on the same database object.

Type: USERSET

Value range: Boolean

- **on** indicates that DDL operations can be performed safely and that no distributed deadlock occurs.
- **off** indicates that DDL operations cannot be performed safely and that distributed deadlocks may occur.

Default value: on

show_acce_estimate_detail

Parameter description: When the GaussDB(DWS) cluster is accelerated (**acceleration_with_compute_pool** is set to **on**), specifies whether the **EXPLAIN** statement displays the evaluation information about execution plan pushdown to computing Node Groups. The evaluation information is generally used by O&M personnel during maintenance, and it may affect the output display of the **EXPLAIN** statement. Therefore, this parameter is disabled by default. The evaluation information is displayed only if the **verbose** option of the **EXPLAIN** statement is enabled.

Type: USERSET

Value range: Boolean

- **on** indicates that the evaluation information is displayed in the output of the **EXPLAIN** statement.
- **off** indicates that the evaluation information is not displayed in the output of the **EXPLAIN** statement.

Default value: off

13.19 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 the 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.

Type: USERSET

Value range: an integer. The minimum value is **0** and 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 the 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).

Type: USERSET

Value range: an integer. The minimum value is **0** and the unit is s.

Default value: 10min

 NOTE

- If the value is **0**, no error is reported when the blocking times out or the transaction is rolled back.
- The value of this parameter must be greater than **gs_clean_timeout**. Otherwise, unnecessary transaction rollback will probably occur due to a block timeout caused by residual transactions that have not been deleted by **gs_clean** on a DN.

13.20 Miscellaneous Parameters

Set the following parameters by referring to [Table 13-1](#).

gin_fuzzy_search_limit

Parameter description: The primary goal of developing GIN indexes was to create support for highly scalable full-text search in GaussDB(DWS), and there are often situations when a full-text search returns a large set of results. Moreover, this situation often occurs when the query contains frequently used words, so that the large result set is not even useful. Because reading many tuples from the disk and sorting them will consume large numbers of resources, this is unacceptable for production. To facilitate controlled execution of such queries, GIN has a configurable soft upper limit on the number of rows returned: configuration parameter **gin_fuzzy_search_limit**. The default value **0** indicates that there is no limit on the returned set. If a non-zero value is set, the returned set is a subset chosen from the whole result set at random.

This parameter is a USERSET parameter.

Value range: 0 to the maximum int value

Default value: 0

enable_cluster_resize

Parameter description: If an SQL statement involves tables belonging to different groups, you can enable this parameter to push the execution plan of the statement to improve performance.

This parameter is a SUSET parameter.

Value range: Boolean

- **on** indicates the execution plan of the statement can be pushed.
- **off** indicates 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.

cstore_insert_mode

Parameter description: Specifies the storage location of data to be imported to an HDFS table. This parameter is needed for operations that involve data import, such as INSERT, UPDATE, COPY, DELTAMERGE, and VACUUM FULL.

This parameter is a USERSET parameter.

Value range: enumerated values

- **AUTO:** The major part of imported data is stored in HDFS, and the tail is stored in the delta table.
- **DELTA:** All the imported data is stored in the delta table.
- **MAIN:** All the imported data is stored in HDFS.

Default value: AUTO

NOTE

You can set other values as the default in the configuration file.

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 avoided when update and delete operations are performed at the same time.

This parameter is a USERSET parameter.

Value range: Boolean

- If this parameter is set to **on**, the delta merge operation internally increases the lock level. In this way, when any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed, an operation can be performed only after the previous one is complete.
- If this parameter is set to **off**, and any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed to data in a row in the

delta table of the HDFS table, errors will be reported during the later operation, and the operation will stop.

Default value: off

ngram_gram_size

Parameter description: Specifies the length of the ngram parser segmentation.

This parameter is a USERSET parameter.

Value range: an integer ranging from 1 to 4

Default value: 2

ngram_grapsymbol_ignore

Parameter description: Specifies whether the ngram parser ignores graphical characters.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Ignores graphical characters.
- **off:** Does not ignore graphical characters.

Default value: off

ngram_punctuation_ignore

Parameter description: Specifies whether the ngram parser ignores punctuations.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Ignores punctuations.
- **off:** Does not ignore punctuations.

Default value: on

zhparser_multi_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Aggregates segments in long words with duality.
- **off:** Does not aggregate segments in long words with duality.

Default value: off

zhparser_multi_short

Parameter description: Specifies whether Zhparser executes long words composite divide.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Performs compound segmentation for long words.
- **off:** Does not perform compound segmentation for long words.

Default value: on

zhparser_multi_zall

Parameter description: Specifies whether Zhparser displays all single words individually.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Displays all single words separately.
- **off:** Does not display all single words separately.

Default value: off

zhparser_multi_zmain

Parameter description: Specifies whether Zhparser displays important single words separately.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Displays important single words separately.
- **off:** Does not display important single words separately.

Default value: off

zhparser_punctuation_ignore

Parameter description: Specifies whether the Zhparser segmentation result ignores special characters including punctuations (\r and \n will not be ignored).

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Ignores all the special characters including punctuations.
- **off:** Does not ignore all the special characters including punctuations.

Default value: on

zhparser_seg_with_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

This parameter is a USERSET parameter.

Value range: Boolean

- **on:** Aggregates segments in long words with duality.
- **off:** Does not aggregate segments in long words with duality.

Default value: off

acceleration_with_compute_pool

Parameter description: Specifies whether to use the computing resource pool for acceleration when a foreign OBS or HDFS table is queried.

This parameter is a USERSET parameter.

Value range: Boolean

- **on** indicates that the query covering the foreign OBS or HDFS table is accelerated based on the cost when the computing resource pool is available.
- **off** indicates that no query is accelerated using the computing resource pool.

Default value: off

behavior_compat_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,).

This parameter is a USERSET parameter.

Value range: a string

Default value: empty

NOTE

- Currently, only [Table 13-4](#) is supported.
- Multiple items are separated by commas (,), for example, **set behavior_compat_options='end_month_calculate,display_leading_zero'**.

Table 13-4 Compatibility configuration items

Configuration Item	Behavior
display_leading_zero	Specifies how floating point numbers are displayed. - If this item is not specified, for a decimal number between -1 and 1, the 0 before the decimal point is not displayed. For example, 0.25 is displayed as .25. - If this item is specified, for a decimal number between -1 and 1, the 0 before the decimal point is displayed. For example, 0.25 is displayed as 0.25.
end_month_calculation	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>select add_months('2018-02-28',3) from dual; 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>select add_months('2018-02-28',3) from dual; 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>.

Configuration Item	Behavior
bind_procedure_searchpath	Specifies the search path of the database object 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.
correct_to_number	Controls the compatibility of the to_number() result. If this item is specified, the value of to_number() is the same as that of pg11. Otherwise, the value is the same as that of Oracle.
unbind_divide_bound	Controls the range check on the result of integer division. If this item is specified, you do not need to check the range of the division result. For example, the result of INT_MIN/(-1) can be INT_MAX+1. If this item is not specified, an out-of-bounds error is reported because the result is greater than INT_MAX.
merge_update_multi	Performs an update if multiple rows are matched for MERGE INTO. If this item is specified, no error is reported if multiple rows are matched. Otherwise, an error is reported (same as Oracle).

Configuration Item	Behavior
return_null_string	Specifies how to display the empty result (empty string '') of the <code>lpad()</code>, <code>rpadd()</code>, <code>repeat()</code>, <code>regexp_split_to_table()</code>, and <code>split_part()</code> functions. In Teradata-compatible mode, this parameter has no impact on the display of an empty result. In Oracle-compatible mode, an empty string will be returned as NULL if this parameter is specified. • If this item is not specified, the empty string is displayed as NULL. <pre>select length(lpad('123',0,'*')) from dual; length ----- (1 row)</pre> • If this item is specified, the empty string is displayed as single quotation marks (''). <pre>select length(lpad('123',0,'*')) from dual; length ----- 0 (1 row)</pre>
compat_concat_variadic	Specifies the compatibility of variadic results of the <code>concat()</code> and <code>concat_ws()</code> functions. If this item is specified and a concat function has a parameter of the variadic type, different result formats in Oracle and Teradata are retained. If this item is not specified and a concat function has a parameter of the variadic type, the result format of Oracle is retained for both Oracle and Teradata.

table_skewness_warning_threshold

Parameter description: Specifies the threshold for triggering a table skew alarm.

This parameter is a SUSET parameter.

Value range: a floating point number ranging from 0 to 1

Default value: 1

table_skewness_warning_rows

Parameter description: Specifies the minimum number of rows for triggering a table skew alarm.

This parameter is a SUSET parameter.

Value range: an integer ranging from 0 to INT_MAX

Default value: 100000

14 Resource Load Management

14.1 Resource Load Management Overview

Resources influencing cluster performance include the memory, CPU, disk I/O, and storage space.

During service operation, system resources may not be fully utilized, and some nodes in a cluster or some resources in the system may become the cluster performance bottleneck. GaussDB(DWS) provides resource management methods to balance system resource usage among tasks.

CPU and memory are computing resources of a server. Centralized control over them prevents resource conflicts of jobs, performs high-priority jobs (such as generating key reports) before others, and isolates user resources.

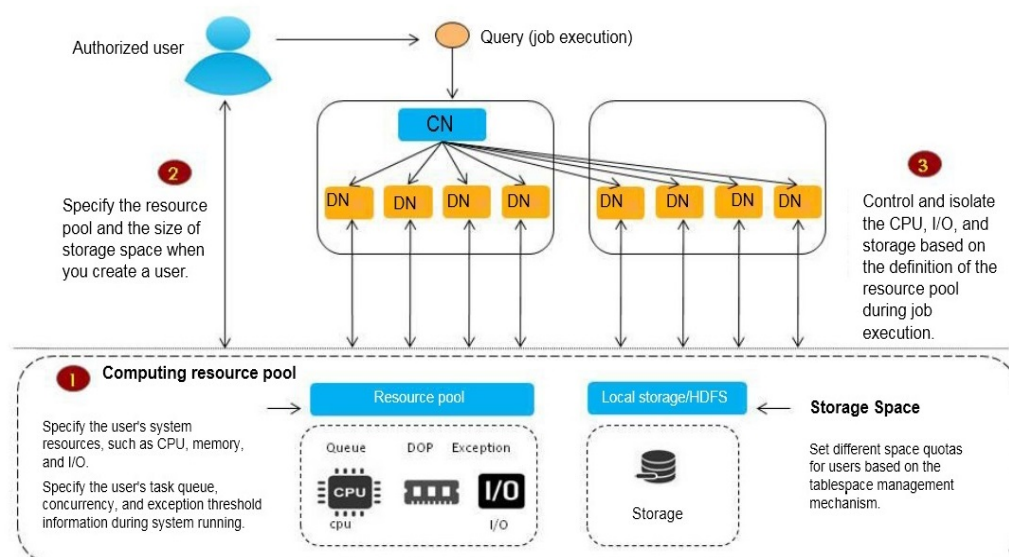
Storage space management is an important feature in the multi-tenant scenario. It limits the storage space quotas for users. GaussDB(DWS) provides a simple syntax interface used for specifying the size of the storage space when you create a user, collecting statistics on and controlling the storage space based on internal logics.

Figure 14-1 shows an example. In the preceding information:

- CPU and I/O resources are managed using a resource pool.
- Memory can be controlled at node and job levels. Memory is managed using database system parameters, GUC parameters, and resource pools.
- Data storage space is specified when you create a user.

NOTE

The resource pool is a basic unit of load management in GaussDB(DWS). It manages system resources (including CPU, I/O, and memory) required by service operations and controls SQL concurrency.

Figure 14-1 Resource load management overview

14.2 Memory Management

14.2.1 Memory Management Overview

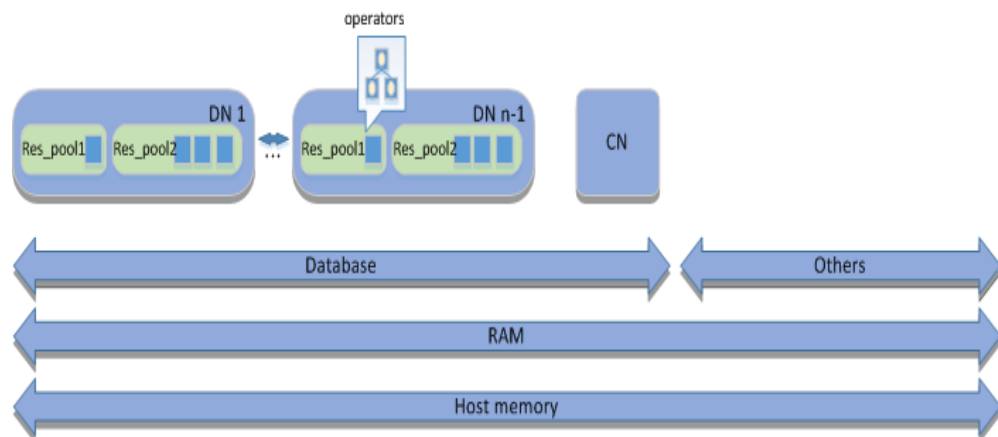
Overview

Sufficient memory is important for GaussDB(DWS) running. Effective utilization of the memory helps improve service query performance.

The resource pool is a collection of system resources managed by GaussDB(DWS), responsible for distributing and managing task resources running in it.

Host Memory

On a GaussDB(DWS) node, the memory on the host is shared among all running processes, including the OS processes, GaussDB(DWS) instance processes, and other application processes. The administrator must monitor the memory shared between GaussDB(DWS) and non-GaussDB(DWS) processes.

Figure 14-2 Memory usage on a GaussDB(DWS) node

- "Host memory" represents the total memory, that is the physical memory RAM on the server. (On a Linux server, the host memory includes the swap cache as well. As GaussDB(DWS) uses precise memory management, here only the physical memory RAM is included in the host memory.) The physical memory RAM is shared by GaussDB(DWS) processes and all other processes running on the host. Some programs may occupy a large volume of resources. You can resolve the problem by adjusting the number of DNs on each node or increasing the physical memory.
- If the node is deployed with a CN, the CN and DNs share the memory occupied by GaussDB(DWS).
- On each DN, the memory occupied by the DN is shared in a resource pool.
- Concurrent SQL statements (represented by small blue blocks in the figure) running in the resource pools share the memory occupied by the resource pool.
- The query optimizer generates an execution plan containing a series of tasks (or operators, as shown in the figure). Task operators, such as table scanning or joining operators, process data records and generate an interim result set. All operators in a query share the memory occupied by the query statement.

Configuring Host Memory

Host memory is the memory shared by all programs on a server. You can increase the physical memory by adding RAM to the server.

Generally, the physical memory and OS configuration are managed by system administrators.

14.2.2 Job-level Memory Control

Context

GaussDB(DWS) can control memory at job level. This helps you more efficiently utilize memory resources than **work_mem** does.

Prerequisites

The cluster has been installed and works properly.

Memory Parameters

You can set **query_mem** for a session to control memory available for jobs.

Procedure

The database administrator can manage the logical memory of the database by taking the following steps:

Step 1 Set **query_mem** to **500MB**.

```
set query_mem='500MB';  
SET
```

Step 2 Execute the job.

----End

NOTICE

If **query_mem** exceeds the upper limit of memory available in the resource pool or is less than 256 MB, **query_mem** will not take effect. Available memory for jobs is still controlled by **work_mem**.

14.3 Resource Load Management Framework Overview

14.3.1 Resource Pools

Context

The core element of resource load management in GaussDB(DWS) is the resource pool, which provides attributes to control memory, I/O, and CPU resources at the bottom layer and to manage and allocate resources based on the priority-based task scheduling mechanism, managing resource load for user services.

Resource pool-oriented load management includes concurrence management and priority-based task scheduling.

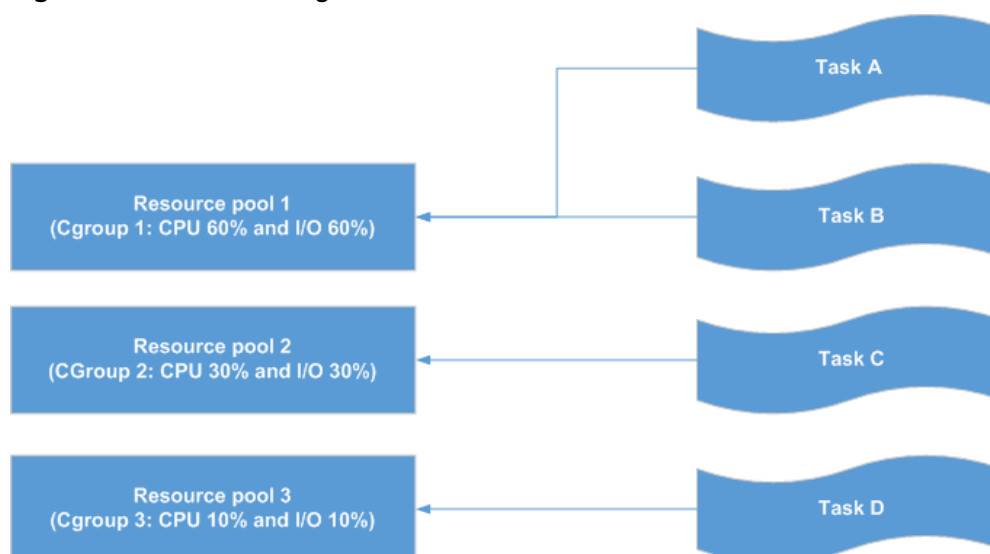
Resource Pools

The resource pool is a resource configuration mechanism for allocating host resources. You can specify resources and concurrency queue attributes for a resource pool, and then set the priority and available resources for a job by binding it to the resource pool.

The database administrator binds the resource pool to its corresponding Cgroup based on different service types.

Resource Pool-Oriented Load Management

Figure 14-3 illustrates how resource loads are managed.

Figure 14-3 Load management

A resource pool allocates resources through its binding Cgroup. The task priority depends on the resource volume allocated to the binding resource pool. Generally, the more resources a resource pool has, the higher priority a binding task possesses, as the task will have more resources for execution.

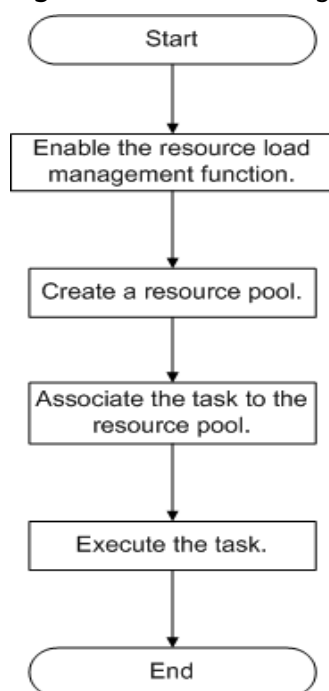
In this figure:

- The priority of task A is higher than that of task C and task D, because the resource pool that task A is bound to has the largest volume of resources.
- Task A and Task B possess the same priority and share system resources allocated to resource pool 1 in equal during competition because the two are in the same resource pool.

To adjust the priority of a task, you can simply change its binding resource pool.

Initialization Process

Figure 14-4 and **Table 14-1** illustrates the details of the operation process in resource load management.

Figure 14-4 Load management workflow**Table 14-1** Load management tasks

Task	Description
Resource Pools	Create a resource pool using the DDL statements provided in GaussDB(DWS), or modify or remove a resource pool using corresponding DDL statements.
Association Tasks	Associate a user to a resource pool to bind the tasks to a specified resource pool before executing the task.
Executing the task	-

default_pool

After the resource load management function is enabled, **default_pool** is automatically created by the system. A session or user not associated with a resource pool is associated with **default_pool**. By default, **default_pool** is bound with the DefaultClass:Medium Cgroup and does not limit the number of concurrent services.

For detailed **default_pool** attributes, see [Table 14-2](#).

Table 14-2 default_pool attributes

Attribute	Value	Description
respool_name	default_pool	Specifies the name of the resource pool.
mem_percent	100	Specifies the maximum memory usage.
cpu_affinity	-1	Specifies the CPU affinity (reserved).
control_group	DefaultClass:Medium	Specifies the Cgroup associated with the resource pool.
active_statements	-1	Specifies the maximum number of concurrent queries allowed by the resource pool. The value -1 indicates that the number of concurrent queries is not limited.
max_dop	1	Concurrency level of execution operators after the SMP is enabled (reserved).
memory_limit	8GB	Specifies the upper memory usage limit (reserved).
parentid	0	Specifies the OID of the parent resource pool.
io_limits	0	Specifies the upper limit of I/Os per second. It is counted by ones in a column-store table and by ten thousands in a row-store table. 0 indicates that the upper limit of I/Os per second is not controlled.
io_priority	None	I/O priority set for jobs that consume many I/O resources. It takes effect when I/O usage reaches 90%. Its values Low , Medium , and High correspond to 25%, 50%, and 80% of the IOPS used by each job when it was triggered, respectively. None indicates the priority is not controlled.

NOTICE

- **default_pool** cannot be modified in GaussDB(DWS).
- For the services associated with **default_pool**, the number of concurrent queries is limited by the **max_active_statements** parameter. **max_active_statements** specifies the maximum number of queries that can be performed on a CN.

Procedure

Creating a resource pool

After resource load management is enabled, the default resource pool is insufficient to address the resource load management requirements of services. Therefore, new resource pools must be used to reallocate system resources to address the requirements.

After connecting to the database as user **dbadmin** or a user with database administrator permissions, you can run SQL statements to create and manage resource pools. For details about the syntax, see CREATE RESOURCE POOL, ALTER RESOURCE POOL, and DROP RESOURCE POOL.

- Create a resource pool associated with the default Cgroup. If you do not specify an associated Cgroup when creating a resource pool, the resource pool is associated with the default Cgroup, which is the **Medium** timeshare Cgroup under the DefaultClass Cgroup.

```
CREATE RESOURCE POOL respool1;
```

If the following information is displayed, the resource pool has been created.

```
CREATE RESOURCE POOL
```

- Create a resource pool associated with the **Rush** timeshare Cgroup.

```
CREATE RESOURCE POOL respool2 WITH (control_group='Rush');
```

If the following information is displayed, the resource pool has been created.

```
CREATE RESOURCE POOL
```

NOTE

- The value of **control_group** is case-sensitive and must be contained in quotation marks ('').
- Only strings indicating Timeshare Cgroups (**Rush**, **High**, **Medium**, or **Low**) can be specified. For example, set **control_group** to **High**. The resource pool is specified to the Timeshare Cgroup whose **control_group** is **High** under DefaultClass.
- Create a resource pool that limits the number of concurrent queries. When creating a resource pool, you can set **active_statements** to specify the maximum number of concurrent tasks associated with the resource pool. If the limit is exceeded, a task queue is implemented to control the number of concurrent tasks.

```
CREATE RESOURCE POOL respool3 WITH (active_statements=5);
```

If the following information is displayed, the resource pool has been created.

```
CREATE RESOURCE POOL
```

 NOTE

1. The default value of **active_statements** is **10**, indicating that there are a maximum of 10 concurrent tasks on a resource pool.
 2. The value range of **active_statements** is from **-1** to **INT_MAX**, where **-1** indicates that the number of concurrent tasks is not limited.
- Create a resource pool that limits available memory. While creating a resource pool, you can set **MEM_PERCENT** to specify the maximum memory available to the resource pool. **MEM_PERCENT** is an integer ranging from 0 to 100.
CREATE RESOURCE POOL *respool4* **WITH (MEM_PERCENT=20);**

If the following information is displayed, the resource pool has been created.

```
CREATE RESOURCE POOL
```

 NOTE

Set the size of the memory used by the resource pool to 20% of the available memory size.

Managing resource pools

After creating resource pools, an administrator often needs to manage resource pools as required, such as adjusting resource pool configurations or deleting obsolete resource pools.

Resource pool management includes:

- Modifying resource pool attributes
 - Modifying Cgroups associated with resource pools
ALTER RESOURCE POOL *respool1* **WITH (control_group="Rush");**
If the following information is displayed, the Cgroup has been modified.
ALTER RESOURCE POOL
 - Modifying the number of concurrent tasks of resource pools
ALTER RESOURCE POOL *respool1* **WITH (ACTIVE_STATEMENTS=15);**
If the following information is displayed, the number of concurrent tasks has been modified.
ALTER RESOURCE POOL
 - Modifying the upper limit on the available memory of resource pools
ALTER RESOURCE POOL *respool1* **WITH (MEM_PERCENT=20);**
If the following information is displayed, the upper limit of the available resource pool memory has been modified.
ALTER RESOURCE POOL
- **DROP RESOURCE POOL** deletes a resource pool.
DROP RESOURCE POOL *respool1*;
DROP RESOURCE POOL

For more usage, see **ALTER RESOURCE POOL**.

Managing Local Concurrent Tasks

The local number of concurrent tasks indicates the maximum number of tasks that can be concurrently executed on one resource pool. The number is specified by **ACTIVE_STATEMENTS**.

Generally, the global number of concurrent tasks must be greater than the sum of local number of concurrent tasks. If the global number of concurrent tasks is less

than the sum of local number of concurrent tasks, the number of tasks concurrently executed is less than the global number of concurrent tasks.

You can specify the number of concurrent tasks of a resource pool either when creating a resource pool or change the number after creating the resource pool.

- To specify the number when creating a resource pool:
CREATE RESOURCE POOL *pool1* **WITH** (**ACTIVE_STATEMENTS**=5);
- To change the number after creating a resource pool:
ALTER RESOURCE POOL *pool1* **WITH** (**ACTIVE_STATEMENTS**=3);

The resource pool uses the method of calculating the concurrent points to calculate the number of concurrent tasks, and the formula is:

Number of points used during task execution: $\text{active\_points} = (\text{query\_mem} / \text{respool\_mem}) \times \text{active\_statements} \times 100$

Total number of points of a resource pool: $\text{total\_points} = \text{active\_statements} \times 100$

Unit points: 100

NOTE

1. The points are used when **query_mem** is not used during task execution.
2. After the total number of points of the resource pool is used up, the queuing operation is triggered in (First In, First Out Queuing) FIFO mode.

Viewing the Current Resource Pool

NOTICE

- Do not use **insert**, **update**, **delete**, or **truncate** in the system catalog **pg_resource_pool** for resource load management. Otherwise, the contents of **pg_resource_pool** queried through different CNs may be inconsistent.
- Do not modify the **memory_limit**, **max_dop**, and **cpu_affinity** attributes of the resource pool.

- View the resource pool bound with a user.

```
SELECT rolrespool FROM PG_AUTHID WHERE rolname = 'rolename';
rolrespool
-----
default_pool
(1 row)
```

- View all the resource pool information of the current cluster.

```
SELECT * FROM PG_RESOURCE_POOL;
respool_name | mem_percent | cpu_affinity | control_group | active_statements | max_dop |
memory_limit | parentid | io_limits | io_priority
-----+-----+-----+-----+-----+-----+
default_pool | 100 | -1 | DefaultClass:Medium | -1 | 1 | 8GB | 0
| 0 | None
(1 row)
```

Viewing the Cgroup Information Associated with a Resource Pool

View the Cgroup information associated with a resource pool. In the following example, **resource_pool_a1** is the name of the resource pool.


```
SELECT * FROM gs_control_group_info('resource_pool_a1');
  name      | class | workload | type | gid | shares | limits | rate | cpucore
-----+-----+-----+-----+-----+-----+-----+-----+-----
class_a:workload_a1 | class_a | workload_a1 | DEFWD | 87 | 30 | 0 | 0 | 0-3
(1 row)
```

Table 14-3 gs_control_group_info attributes

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
type	DEFWD	Cgroup type (Top, CLASS, BAKWD, DEFWD, and 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
cpucore	0-3	CPU core count

14.3.2 Association Tasks

Context

After allocating system resources using resource pools, you must associate tasks with resource pools for service load management.

After a resource pool is associated with a user, all the tasks of the user are automatically associated with the resource pool.

Procedure

Step 1 Create a resource pool **respool** associated with the default Cgroup.

```
CREATE RESOURCE POOL respool;
CREATE RESOURCE POOL
```

NOTE

You can associate the resource pool with a user-defined Cgroup as required. For details, see [Resource Pools](#).

Step 2 Create user **user1** and associate the user with the **respool** resource pool.

```
CREATE USER user1 WITH RESOURCE POOL 'respool' password 'Bigdata@123';
CREATE USER
```

Step 3 Run **job.sql** as user **user1**.

```
gsql -U user1 -W Bigdata@123 -d postgres -p 8000 -f job.sql
```

Alternatively, connect to the related database as user **user1** and run the task.

```
gsql -U user1 -W Bigdata@123 -d postgres -p 8000
```

----End

14.4 Priority Scheduling

14.4.1 CPU Priority Scheduling

Priority scheduling can be based on CPU or I/O usage. The former puts new tasks in a resource pool based on the priority of associated cgroups, and the latter suspends I/O requests in the database. This section mainly describes CPU priority scheduling.

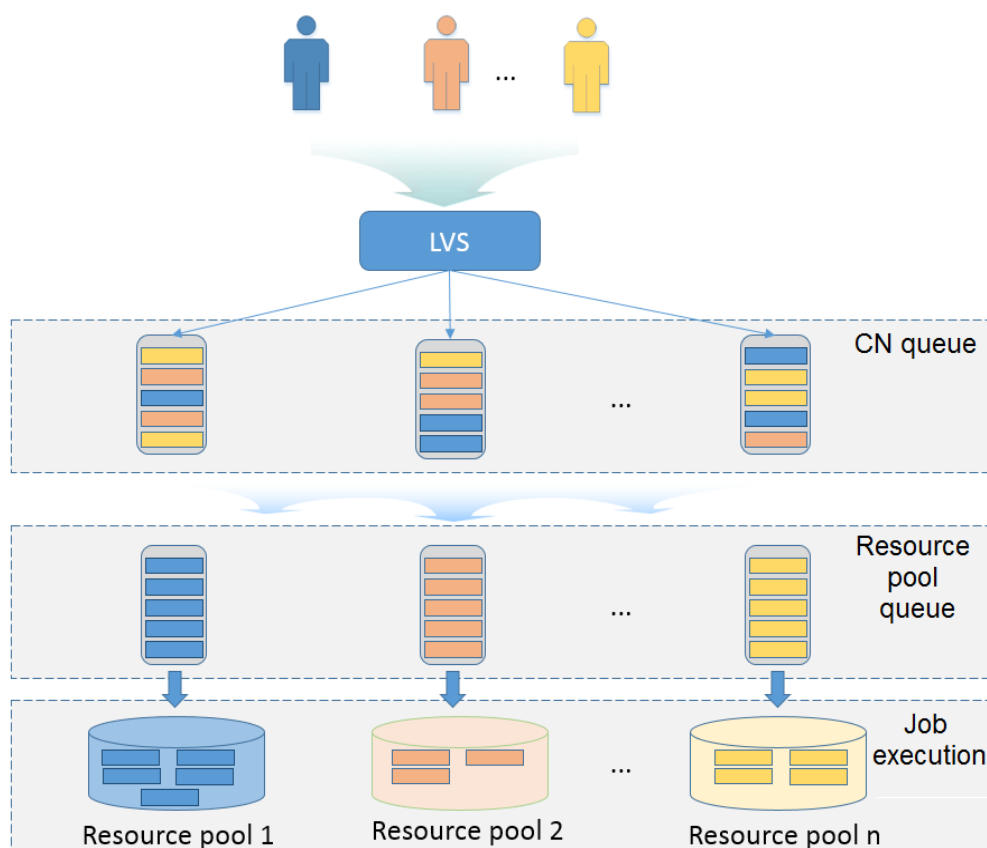
Prerequisites

Resource load management has been enabled.

Context

After the number of concurrent tasks of a resource pool reaches the upper limit, new tasks enter the waiting queue of the resource pool.

[Queue in load management](#) shows the queue mechanism for executing waiting tasks when resource load management is enabled.

Figure 14-5 Queue in load management

Users' tasks enter queues of CNs, queues of resource pools, and then resource pools for execution. The process is described as follows:

1. Users' tasks are distributed by the Linux Virtual Server (LVS) to queues of CNs. Alternatively, tasks are directly distributed by users to the queues. On any CN, tasks are processed in the first-in-first-out (FIFO) principle, which means that a task entering the queue of a CN earlier than another task enters the following step before the other task.
2. In the queue of any CN, a task is distributed to the queue of the resource pool corresponding to the task according to the following rules: based on the following rules:
 - The tasks in the queue of a CN are distributed in the FIFO principle.
 - A task is distributed to the queue of the resource pool bound with the user who initiates the task. If no resource pool is bound with the user, the task is distributed to the queue of the default resource pool **default_pool**.
3. The tasks in the queue of a resource pool are executed in the FIFO principle. If the idle resources in a resource pool meet the requirement of the first task (counted from bottom to top) in the queue of the resource pool, the task enters the resource pool for execution.

Procedure

- Adjust resource pool priorities.

If the resource proportion of a resource pool changes, change the priority of the resource pool accordingly. You can change the priority by modifying attributes in the resource pool.

```
ALTER RESOURCE POOL respool2 WITH(CONTROL_GROUP = "Rush");  
ALTER RESOURCE POOL
```

NOTE

The preceding statement changes the attribute of the Cgroup associated with a resource pool to **Rush**.

- Adjust the priority of the current user. You can change the priority of a user by changing the resource pool it binds to. For example:

```
ALTER USER user1 WITH RESOURCE POOL 'respool2';  
ALTER USER
```

Where **user1** is the user whose priority is to be changed, and **respool2** is the resource pool it is to be bound to. You need the sysadmin permission to change the priority.

- Adjust the priority of the current session.

You can change the Cgroup bound with the current session to adjust the priorities of the tasks in the session. For example:

```
SET CGROUP_NAME="Rush";  
SET
```

Query for the Cgroup associated with the current session.

```
SHOW CGROUP_NAME;
```

- Manually changing the positions of tasks in queues.

If the priority of a task must be increased or a task must be executed at an earlier time, an administrator can change the position of the task in the queue.

- Query for the PID of the task in the queue of a CN.

```
SELECT * FROM pg_session_wlmstat where status = 'pending';
```

- Move the task to the first position of the queue.

```
SELECT pg_wlm_jump_queue(threadid);
```

- Change the priority of the task to **High** during the task execution. You can query the **pg_session_wlmstat** view for the thread ID of the task.

```
SELECT gs_wlm_switch_cgroup(threadid, 'High');
```

NOTE

Only the administrator can adjust the priorities of tasks being executed.

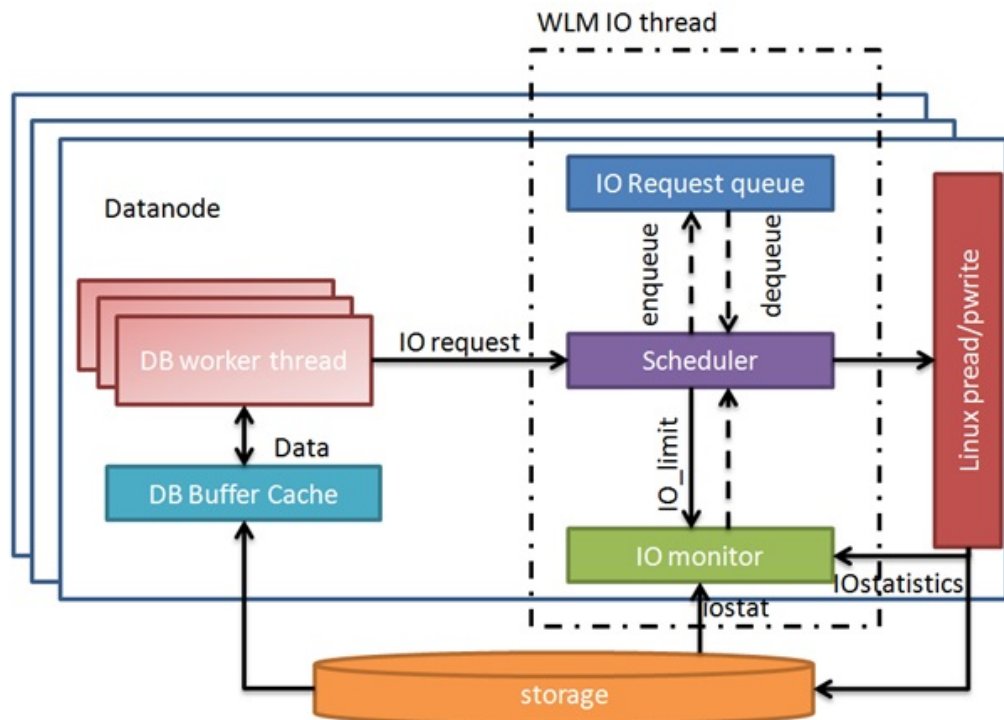
14.4.2 I/O Priority Scheduling

Context

Control I/O resources for data import or complex queries.

I/O resources can be managed based on priority or upper limits at session- or user-level.

Figure 14-6 illustrates how I/O resources are managed.

Figure 14-6 Managing I/O resources

I/O resource management complies with the following rules:

- Not all statement I/O is controlled. The write I/O of only **INSERT INTO**, **SELECT**, **COPY FROM**, and **CREATE TABLE AS** statements can be controlled. Read I/O will be controlled if the amount of data to be scanned on a single DN exceeds 500 MB. The data volume is estimated by the optimizer. **VACUUM FULL**, **CREATE INDEX**, **CLUSTER TABLE USING INDEX**, **ANALYZE**, and **MERGE INTO** are read/write operations and will possibly trigger I/O control under certain conditions.
- You need to manually set GUC parameter or resource pool settings to enable I/O control.
- The GUC parameters **io_priority** and **io_limits** can control I/O based on priority and upper limit, respectively, at session level.
- You can set **io_limits** or **io_priority**, and associate a user with a resource pool to implement control I/O at user level.
- **io_limits** controls the maximum number of I/O requests that can be triggered per second. The default value is **0**, indicating no control. The number of requests is counted by ones for column storage and by thousands for row storage.
- When the disk usage reaches 90%, the I/O control specified by **io_priority** is triggered. The priority can be **High**, **Medium**, or **Low**, which indicate that 80%, 50%, or 20% of I/O triggered by jobs are controlled. The default value is **None**, indicating no control.

 NOTE

- According to the buffer policy, read/write I/O requests are controlled for row storage during buffer read/write instead of disk read/write. The unit of **io_limits** is ten thousand times per second. The value can be set to around **100**. You are advised to use **io_priority** for row storage, so that I/O resources can be fully utilized when they are sufficient.
- The mechanisms for querying and importing data in column-store storage are different from those in row storage. The value of **io_limits** can be set to around **100**. Values much larger may fail to control I/O.

I/O Resource Management View

- **pg_session_iostat** monitors I/O resources at session level.

```
select * from pg_session_iostat;
```
- **pg_user_iostat** monitors I/O resources at user level.

```
select * from pg_user_iostat('username');
```

14.5 Storage Space Management

Scenarios

Users using too much storage space may result in service blocking. To prevent this problem, you need to enhance the control over the storage space.

Storage Space Management

- User space limit

Procedure

Assume that you need to set the maximum space of user **user1** to 100 GB.

Step 1 Create user **user1**.

```
CREATE USER user1 PASSWORD 'Bigdata@123' PERM SPACE '100G';  
CREATE USER
```

Step 2 Insert data as user **user1**.

----End

 NOTE

1. The CN periodically obtains and checks space information from a DN. The interval is 10s.
2. When detecting that the required space exceeds the current available space, the CN cancels all data inserting jobs and records the following information in the log file in the **pg_log** directory: "cancel thread <threadid> for space limit exceed."

14.6 Resource Monitoring

14.6.1 Querying User Resources

Context

In the multi-tenant management framework, you can query the real-time or historical usage of all user resources (including memory, CPU cores, storage space, temporary space, and I/Os).

NOTE

- User real-time resource views/functions: [PG_TOTAL_USER_RESOURCE_INFO](#) and [GS_WLM_USER_RESOURCE_INFO](#); user historical resource table: [GS_WLM_USER_RESOURCE_HISTORY](#).
- In the preceding views and tables, the value of **used_cpu** indicates the CPU usage of a user's resource pool. The resource pool records only the CPU usage of complex jobs.
- In the preceding views and tables, I/O-related resource statistics only record I/O read and write data of complex jobs executed by users.
- When there are a large number of users and a large cluster, querying such real-time views will cause network latency due to the real-time communication overhead between CNs and DN.
- User memory and CPU monitoring does not apply to simple jobs or administrator jobs.

Procedure

- Query all users' resource quotas and real-time resource usage.

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
```

The result view is as follows:

username	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed
perfadm	0	17250	0	0	0	-1	0	-1	0	0	0	0	0	0	0	0
usern	0	17250	0	48	0	-1	0	-1	0	0	0	0	0	0	0	-1
userg	34	15525	23.53	48	0	-1	0	-1	0	0	0	0	0	0	0	0
-1 814955731 8001	-1	6111952	1145864	763994	143233	42678										
userg1	34	13972	23.53	48	0	-1	0	-1	0	0	0	0	0	0	0	0
-1 814972419 8007	-1	6111952	1145864	763994	143233	42710										

(4 rows)

The I/O resource monitoring fields (**read_kbytes**, **write_kbytes**, **read_counts**, **write_counts**, **read_speed**, and **write_speed**) can be available only when the GUC parameter `enable_user_metric_persistent` is enabled.

For details about each column, see [PG_TOTAL_USER_RESOURCE_INFO](#).

- Query a user's resource quota and real-time resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
```

The query result is as follows:

userid	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed

```
16407 |      18 |      1655 |      6 |      19 |      13787176 |      -1 |      0 |      -1
|      0 |      -1 |      0 |      0 |      0 |      0 |      0 |      0
(1 row)
```

- Query all users' resource quotas and historical resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_HISTORY;
```

The query result is as follows:

username	timestamp	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed
usern	2020-01-08 22:56:06.456855+08	0	17250	0	48	0	5710	0	0	0	0	0	0	0	0	0	0
userg	2020-01-08 22:56:06.458659+08	0	15525	33.48	48	0	0	0	0	0	0	0	0	0	0	0	0
userg1	2020-01-08 22:56:06.460252+08	0	13972	33.48	48	0	0	0	0	0	0	0	0	0	0	0	0

For the system catalog in [GS_WLM_USER_RESOURCE_HISTORY](#), data in the [PG_TOTAL_USER_RESOURCE_INFO](#) view is periodically saved to historical tables only when the GUC parameter `enable_user_metric_persistent` is enabled.

For details about each column, see [GS_WLM_USER_RESOURCE_HISTORY](#).

- Query the real-time I/O control resource usage of a specific user. (The value in the view indicates the I/O control resource usage instead of the actual I/O read/write data.)

```
SELECT * FROM pg_user_iostat('username');
```

The query result is as follows:

userid	min_curr_iops	max_curr_iops	min_peak_iops	max_peak_iops	io_limits	io_priority
10	0	0	0	0	0	None

(1 row)

14.6.2 Monitoring Memory Resources

Monitoring the Memory

GaussDB(DWS) provides a view for monitoring the memory usage of the entire cluster.

Query the `pgxc_total_memory_detail` view as a user with `sysadmin` permissions.

```
SELECT * FROM pgxc_total_memory_detail;
```

If the following error message is returned during the query, enable the memory management function.

```
SELECT * FROM pgxc_total_memory_detail;
ERROR: unsupported view for memory protection feature is disabled.
CONTEXT: PL/pgSQL function pgxc_total_memory_detail() line 12 at FOR over EXECUTE statement
```

Monitoring the Shared Memory

You can query the context information about the shared memory on the `pg_shared_memory_detail` view.

```
SELECT * FROM pg_shared_memory_detail;
contextname | level | parent | totalsize | freesize | usedsize
```


ProcessMemory	0		24576	9840	14736	
Workload manager memory context	1	ProcessMemory		2105400	7304	2098096
wlm collector hash table	2	Workload manager memory context		8192	3736	4456
Resource pool hash table	2	Workload manager memory context		24576	15968	8608
wlm cgroup hash table	2	Workload manager memory context		24576	15968	8608
(5 rows)						

This view lists the context name of the memory, level, the upper-layer memory context, and the total size of the shared memory.

In the database, GUC parameter **memory_tracking_mode** is used to configure the memory statistics collecting mode, including the following options:

- **none:** The memory statistics collecting function is not enabled.
- **normal:** Only memory statistics is collected in real time and no file is generated.
- **executor:** The statistics file is generated, containing the context information about all allocated memory used by the execution layer.

If the parameter is set to **executor**, cvs files are generated under the **pg_log** directory of the DN process. The file names are in the format of **memory_track_<DN name>_query_<queryid>.csv**. The information about the operators executed by the postgres thread of the executor and all stream threads are input in this file during task execution.

The instance is built with a file content similar to the following:

```
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 40K, 602K, 23
1, 3, CStoreScan_29360131_25, 0, ExecutorState, 1, 265K, 554K, 23
2, 128, cstore scan per scan memory context, 1, CStoreScan_29360131_25, 2, 24K, 24K, 23
3, 127, cstore scan memory context, 1, CStoreScan_29360131_25, 2, 264K, 264K, 23
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_25, 2, 31K, 31K, 23
5, 2, VecPartIterator_29360131_24, 0, ExecutorState, 1, 16K, 16K, 23
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 24K, 1163K, 20
1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20
2, 20, cstore scan per scan memory context, 1, CStoreScan_29360131_22, 2, 476K, 476K, 20
3, 19, cstore scan memory context, 1, CStoreScan_29360131_22, 2, 264K, 264K, 20
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_22, 2, 23K, 23K, 20
5, 2, VecPartIterator_29360131_21, 0, ExecutorState, 1, 16K, 16K, 20
```

The columns include the output SN, SN of the memory allocation context within the thread, name of the current memory context, output SN of the parent memory context, name of the parent memory context, tree layer No. of the memory context, peak memory used by the current memory context, peak memory used by the current memory context and all its child memory contexts, and plan node ID of the query where the thread is executed.

In this example, the record "1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20" represents the following information about Explain Analyze:

- **CStoreScan_29360131_22** indicates the CstoreScan operator.
- **1122K** indicates the peak memory used by the CstoreScan operator.
- **fullexec:** The generated file includes the information about all memory contexts requested by the execution layer.

When the parameter is set to **fullexec**, the output information is the same as that of **executor**. The only difference is that in **fullexec** mode, some memory context allocation information may be included, because the information about all memory applications (no matter succeeded or not) is printed. As only the memory application information is recorded, the peak memory used by the memory context is recorded as **0**.

14.6.3 Instance Resource Monitoring

GaussDB(DWS) provides system catalogs for monitoring the resource usage of CNs and DNs (including memory, CPU usage, disk I/O, process physical I/O, and process logical I/O), and system catalogs for monitoring the resource usage of the entire cluster.

For details about the system catalog **GS_WLM_INSTANCE_HISTORY**, see [GS_WLM_INSTANCE_HISTORY](#).

NOTE

Data in the system catalog **GS_WLM_INSTANCE_HISTORY** is distributed in corresponding instances. CN monitoring data is stored in the CN instance, and DN monitoring data is stored in the DN instance. The DN has a standby node. When the primary DN is abnormal, the monitoring data of the DN can be restored from the standby node. However, a CN has no standby node. When a CN is abnormal and then restored, the monitoring data of the CN will be lost.

Procedure

- Query the latest resource usage of the current instance.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY ORDER BY TIMESTAMP DESC;
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
dn_6015_6016	2020-01-10 17:29:17.329495+08	0	14570	8982	662.923	99.9601	697666	93655.5	183104	30082	285659	30079	357717	37667
dn_6015_6016	2020-01-10 17:29:07.312049+08	0	14578	8974	883.102	99.9801	756228	81417.4	189722	30786	285681	30780	358103	38584
dn_6015_6016	2020-01-10 17:28:57.284472+08	0	14583	8969	727.135	99.9801	648581	88799.6	177120	31176	252161	31175	316085	39079
dn_6015_6016	2020-01-10 17:28:47.256613+08	0	14591	8961	679.534	100.08	655360	169962	179404	30424	242002	30422	303351	38136

- Query the resource usage of the current instance during a specified period.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY WHERE TIMESTAMP > '2020-01-10' AND TIMESTAMP < '2020-01-11' ORDER BY TIMESTAMP DESC;
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
dn_6015_6016	2020-01-10 17:29:17.329495+08	0	14570	8982	662.923	99.9601	697666	93655.5	183104	30082	285659	30079	357717	37667
dn_6015_6016	2020-01-10 17:29:07.312049+08	0	14578	8974	883.102	99.9801	756228	81417.4	189722	30786	285681	30780	358103	38584
dn_6015_6016	2020-01-10 17:28:57.284472+08	0	14583	8969	727.135	99.9801	648581	88799.6	177120	31176	252161	31175	316085	39079
dn_6015_6016	2020-01-10 17:28:47.256613+08	0	14591	8961	679.534	100.08	655360	169962	179404	30424	242002	30422	303351	38136

- To query the latest resource usage of a cluster, you can invoke the **pgxc_get_wlm_current_instance_info** stored procedure on the CN.

```
SELECT * FROM pgxc_get_wlm_current_instance_info('ALL');
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts

write_counts

coordinator2	2020-01-14 21:58:29.290894+08		0		12010		278		16.0445	7.19561
184.431	27959.3		0		10		0		0	
coordinator3	2020-01-14 21:58:27.567655+08		0		12000		288		.964557	3.40659
332.468	3375.02		26		13		0		0	
datanode1	2020-01-14 21:58:23.900321+08		0		11899		389		1.17296	3.25
329.6	2870.4		28		8		13		3	18
datanode2	2020-01-14 21:58:32.832989+08		0		11904		384		17.948	8.52148
214.186	25894.1		28		10		13		3	18
datanode3	2020-01-14 21:58:24.826694+08		0		11894		394		1.16088	3.15
2868.8	25		10		13		3		18	6
coordinator1	2020-01-14 21:58:33.367649+08		0		11988		300		9.53286	10.05
43.2	55232		0		0		0		0	
coordinator1	2020-01-14 21:58:23.216645+08		0		11988		300		1.17085	3.21182
324.729	2831.13		8		13		0		0	
(7 rows)										

- To query historical resource usage of a cluster, you can invoke the **pgxc_get_wlm_current_instance_info** stored procedure on the CN.

```
SELECT * FROM pgxc_get_wlm_history_instance_info('ALL', '2020-01-14 21:00:00', '2020-01-14 22:00:00', 3);
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
coordinator2	2020-01-14 21:50:49.778902+08		0		12020		268		.127371		.789211			
15.984	3994.41		0		0		0		0		0			
coordinator2	2020-01-14 21:53:49.043646+08		0		12018		270		30.2902		8.65404			
276.77	16741.8		3		1		0		0		0			
coordinator2	2020-01-14 21:57:09.202654+08		0		12018		270		.16051		.979021			
59.9401	5596		0		0		0		0		0			
coordinator3	2020-01-14 21:38:48.948646+08		0		12012		276		.0769231		.00999001			
0	35.1648		0		1		0		0		0			
coordinator3	2020-01-14 21:40:29.061178+08		0		12012		276		.118421		.0199601			
0	970.858		0		0		0		0		0			
coordinator3	2020-01-14 21:50:19.612777+08		0		12010		278		24.411		11.7665			
8.78244	44641.1		0		0		0		0		0			
datanode1	2020-01-14 21:49:42.758649+08		0		11909		379		.798776		8.02			
51.2	20924.8		0		0		0		0		0			
datanode1	2020-01-14 21:49:52.760188+08		0		11909		379		23.8972		14.1			
0	74760		0		0		0		0		0			
datanode1	2020-01-14 21:50:22.769226+08		0		11909		379		39.5868		7.4		0	
19760.8	0		0		0		0		0		0			
datanode2	2020-01-14 21:58:02.826185+08		0		11905		383		.351648		.32			
20.8	504.8		0		0		0		0		0			
datanode2	2020-01-14 21:56:42.80793+08		0		11906		382		.559748		.04		0	
326.4	0		0		0		0		0		0			
datanode2	2020-01-14 21:45:21.632407+08		0		11901		387		12.1313		4.55544			
3.1968	45177.2		0		0		0		0		0			
datanode3	2020-01-14 21:58:14.823317+08		0		11898		390		.378205		.99			
48	23353.6		0		0		0		0		0			
datanode3	2020-01-14 21:47:50.665028+08		0		11901		387		1.07494		1.19			
0	15506.4		0		0		0		0		0			
datanode3	2020-01-14 21:51:21.720117+08		0		11903		385		10.2795		3.11			
0	11031.2		0		0		0		0		0			
coordinator1	2020-01-14 21:42:59.121945+08		0		12020		268		.0857143		.0699301			
0	6579.02		0		0		0		0		0			
coordinator1	2020-01-14 21:41:49.042646+08		0		12020		268		20.9039		11.3786			
6042.76	57903.7		0		0		0		0		0			
coordinator1	2020-01-14 21:41:09.007652+08		0		12020		268		.0446429		.03996			
0	1109.29		0		0		0		0		0			
(18 rows)														

14.6.4 Real-Time TopSQL

When a database user performs a query job, the system provides real-time resource monitoring views at the query- and operator-levels to query SQL queries in the **Active** state for the real-time TopSQLs whose execution cost is greater than **resource_track_cost**. The views record the resource usage (including memory, disk, CPU time, and I/O) and performance alarms during job execution. Based on the recorded information, you can evaluate whether the query has performance bottlenecks and whether it affects the cluster performance.

By default, query-level resource monitoring is enabled. To enable resource monitoring at both the query and operator levels, set **resource_track_level** to **operator**. The following jobs support real-time resource monitoring:

- Jobs whose execution cost estimated by the optimizer is greater than or equal to **resource_track_cost**. You can run the **EXPLAIN** statement to query for the execution cost of a statement.
- Jobs of the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **CREATE TABLE AS**, **EXPLAIN ANALYZE**, and **EXPLAIN PERFORMANCE** statements. A query-level view also supports the **FETCH** statement. In a query-level view, jobs of the **FETCH** statement are also monitored.

The real-time resource monitoring view is described in the following table. You can query such a view in the same way as querying database tables. Common users can only query views with the prefix **gs**.

Table 14-4 Real-time resource monitoring view

Monitoring Level	Monitored Node	Query View	View Description
Query	Current CN	GS_WLM_SESSION_STATISTICS	Queries for the real-time resources on the current CN.
	All CNs	PGXC_WLM_SESSION_STATISTICS	Queries for the real-time resources on all CNs.
Operator	Current CN	GS_WLM_OPERATOR_STATISTICS	Queries for the real-time resources of job operators on the current CN.
	All CNs	PGXC_WLM_OPERATOR_STATISTICS	Queries for the real-time resources of job operators on all CNs.

 NOTE

- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- When you query this type of views, there will be network latency, because the views obtain resource usage in real time.
- If instance fault occurs, some SQL statement information may fail to be recorded in real-time resource monitoring views.
- SQL statements are recorded in real-time resource monitoring views as follows:
 - DDL statements are not recorded, such as **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE**, and **VACUUM**.
 - DML statements are recorded, such as **SELECT**, **INSERT**, **UPDATE**, and **DELETE**.
 - Statements in functions and stored procedures and statements used for calling functions and stored procedures are recorded. Statements in loop bodies (if any) of functions and stored procedures are not recorded.
 - Statements in anonymous blocks are not recorded.
 - Statements in transaction blocks are recorded. Statements in loop bodies (if any) of transaction blocks are not recorded.
 - Cursor statements are recorded.

Typical Queries for Real-Time Top SQLs

The query for statements executed on the current CN is used as an example to describe some typical SQL statements used for querying for Top SQLs in terms of real-time resource usage and performance. To query for the SQL statements executed on all CNs, replace the prefix **gs** of the view name in the query statement with **pgxc**.

- Query level. By default, the first column in each query statement provided in the example is **query** (that is, the SQL statement to be queried).
 - Query for the start time, end time, execution time, and execution status of the SQL statement. The time unit is ms.

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_history ORDER BY start_time DESC;
```
 - Query for the maximum memory peak of the SQL statement among all DN, average memory usage during statement execution, and the memory usage skew ratio across DN. The memory unit is MB.

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM gs_wlm_session_statistics ORDER BY start_time DESC;
```
 - Query for the spill of the SQL statement to each DN, maximum data spill volume among all DN, average data spill volume among all DN, and spill skew ratio across DN. The data volume unit is MB.

```
SELECT query,spill_info,max_spill_size,average_spill_size,spill_skew_percent FROM gs_wlm_session_statistics ORDER BY start_time DESC;
```
 - Query for the maximum execution time of the SQL statements among all DN and the execution time skew ratio across DN. The time unit is ms.

```
SELECT query,max_dn_time,dntime_skew_percent FROM gs_wlm_session_statistics ORDER BY start_time DESC;
```
 - Query for the maximum CPU time of the SQL statement among all DN, the total CPU time on all DN, and the CPU time skew across DN. The time unit is ms.

```
SELECT query,max_cpu_time,total_cpu_time,cpu_skew_percent FROM gs_wlm_session_statistics ORDER BY start_time DESC;
```

- Query for the maximum IOPS peak of the SQL statement among all DNs, the average IOPS peak among all DNs, and the I/O skew ratio across DNs. The I/Os are counted by ones for column storage and by 10 thousands for row storage.

```
SELECT query,max_peak_iops,average_peak_iops,iops_skew_percent FROM  
gs_wlm_session_statistics ORDER BY start_time DESC;
```
- Operator level. By default, the first column in each query statement provided in the example is **plan_node_name** (that is, the operator name matching **plan_node_id**).
 - Query for the start time for the operator to process data, total data processing duration (unit: ms) of the operator, and operator execution status.

```
SELECT plan_node_name,start_time,duration,status FROM gs_wlm_operator_statistics ORDER BY  
start_time DESC;
```
 - Query for the maximum memory peak of the operator among all DNs, average memory peak among all DNs, and memory usage skew ratio across DNs. The memory unit is MB.

```
SELECT plan_node_name,max_peak_memory,average_peak_memory,memory_skew_percent  
FROM gs_wlm_operator_statistics ORDER BY start_time DESC;
```
 - Query for the maximum data spill volume of the operator among all DNs, average data spill volume among all DNs, and spill skew ratio across DNs. The data volume unit is MB.

```
SELECT plan_node_name,max_spill_size,average_spill_size,spill_skew_percent FROM  
gs_wlm_operator_statistics ORDER BY start_time DESC;
```
 - Query for the maximum execution time of the operator among all DNs, total execution time on all DNs, and execution time skew ratio across DNs. The time unit is ms.

```
SELECT plan_node_name,max_cpu_time,total_cpu_time,cpu_skew_percent FROM  
gs_wlm_operator_statistics ORDER BY start_time DESC;
```

Precautions

- Jobs in a redistribution process are not monitored.
- Prefixes **gs** and **pgxc** indicate real-time views showing single CN information and those showing cluster information, respectively. Common users can query only the views with the prefix **gs**.
- When you query real-time views, there will be network latency, because the views obtain resource usage in real time.
- When you query views in [Table 14-4](#), network latency may be caused by the views to obtain real-time resource usage.
- If you only need to query for the real-time CPU information, use the [GS_SESSION_CPU_STATISTICS](#) view; if you only need to query for the real-time memory information, use the [GS_SESSION_MEMORY_STATISTICS](#) view. The [GS_WLM_SESSION_STATISTICS](#) view returns a query result containing both [GS_SESSION_CPU_STATISTICS](#) and [GS_SESSION_MEMORY_STATISTICS](#).

14.6.5 Historical TopSQL

After the execution of a job is complete, you can retrieve its historical information, including the resource usage (such as the usage of memory, disk, CPU time, and I/O), running status (such as error, terminated, and exception), and performance alarms. You can use historical resource monitoring views at the query- and operator-levels to query for historical TopSQLs whose execution cost is greater than [resource_track_cost](#).

By default, query-level resource monitoring is enabled. To enable resource monitoring at both the query and operator levels, set [resource_track_level](#) to **operator**. The following jobs support historical resource monitoring:

- Jobs whose execution cost estimated by the optimizer is greater than or equal to [resource_track_cost](#). You can run the **EXPLAIN** statement to query for the execution cost of a statement.
- Jobs whose execution time recorded in the real-time resource monitoring view is greater than or equal to [resource_track_duration](#).

The historical resource monitoring view is described in the following table. You can query such a view in the same way as querying database tables. Common users can only query views with the prefix **gs**.

Table 14-5 Historical resource monitoring view

Monitoring Level	Monitored Node	Query View	View Description
Query	Current CN	GS_WLM_SESSION_HISTORY	Queries for the load of an executed job.
	All CNs	PGXC_WLM_SESSION_HISTORY	Queries for the load of an executed job.
Operator	Current CN	GS_WLM_OPERATOR_HISTORY	Queries for the operator resource information of an executed job.
	All CNs	PGXC_WLM_OPERATOR_HISTORY	Queries for the operator resource information of an executed job.

 NOTE

- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- You can query the [gs_wlm_session_info](#), [gs_wlm_operator_info](#), [pgxc_wlm_session_info](#), and [pgxc_wlm_operator_info](#) views only after connecting to the **postgres** database.
- If instance fault occurs, some SQL statement information may fail to be recorded in historical resource monitoring views.
- The SQL statements that can be recorded in historical resource monitoring views are the same as those recorded in real-time resource monitoring views. For details, see [SQL statements recorded in real-time resource monitoring views](#).

The resource monitoring information stored in the **history** view (for example, [Table 14-5](#)) is archived to the **info** view (for example, [Table 14-6](#)) at an interval of 3 minutes. After being archived, the records are deleted from the **history** view. The **info** view can be queried only when the postgres database is connected.

Table 14-6 Archiving view of historical resource monitoring

Monitoring Level	Monitored Node	Query View	View Description
Query	Current CN	GS_WLM_SESSION_INFO	Queries for the load of an executed job.
	All CNs	PGXC_WLM_SESSION_INFO	Queries for the load of an executed job.
Operator	Current CN	GS_WLM_OPERATOR_INFO	Queries for the operator resource information of an executed job.
	All CNs	PGXC_WLM_OPERATOR_INFO	Queries for the operator resource information of an executed job.

 NOTE

If **enable_resource_record** is set to **on**, storage space expansion may occur and thereby slightly affects the performance. Therefore, set it to **off** if record archiving is unnecessary.

Typical Queries for Historical Top SQLs

The query for statements whose execution is complete on the current CN within the last 3 minutes is used as an example. To query for the SQL statements executed on all CNs, replace the prefix **gs** of the view name in the query statement with **pgxc**. To query for the SQL statements whose execution is complete 3 minutes ago, replace the suffix **history** of the view name in the query statement with **info**.

- Query level. By default, the first column in each query statement provided in the example is **query** (that is, the SQL statement to be queried).
 - Query for the start time, end time, execution time, and execution status of the SQL statement. The time unit is ms.

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_history ORDER BY start_time DESC;
```
 - Query for the maximum memory peak of the SQL statement among all DN, average memory usage during statement execution, and the memory usage skew ratio across DNs. The memory unit is MB.

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM gs_wlm_session_history ORDER BY start_time DESC;
```
 - Query for the spill of the SQL statement to each DN, maximum data spill volume among all DNs, average data spill volume among all DNs, and spill skew ratio across DNs. The data volume unit is MB.

```
SELECT query,spill_info,max_spill_size,average_spill_size,spill_skew_percent FROM gs_wlm_session_history ORDER BY start_time DESC;
```
 - Query for the maximum execution time of the SQL statements among all DNs, average execution time of the SQL statement among all DNs, and the execution time skew ratio across DNs. The time unit is ms.

- ```
SELECT query,max_dn_time,average_dn_time,dntime_skew_percent FROM
gs_wlm_session_history ORDER BY start_time DESC;
```
- Query for the maximum CPU time of the SQL statement among all DNs, the total CPU time on all DNs, and the CPU time skew across DNs. The time unit is ms.  

```
SELECT query,max_cpu_time,total_cpu_time,cpu_skew_percent FROM gs_wlm_session_history
ORDER BY start_time DESC;
```
  - Query for the maximum IOPS peak of the SQL statement among all DNs, the average IOPS peak among all DNs, and the I/O skew ratio across DNs. The I/Os are counted by ones for column storage and by 10 thousands for row storage.  

```
SELECT query,max_peak_iops,average_peak_iops,iops_skew_percent FROM
gs_wlm_session_history ORDER BY start_time DESC;
```
  - Operator level. By default, the first column in each query statement provided in the example is **plan\_node\_name** (that is, the operator name matching **plan\_node\_id**).
    - Query for the start time for the operator to process data, total data processing duration of the operator, and operator execution status. The time unit is ms.  

```
SELECT plan_node_name,start_time,duration FROM gs_wlm_operator_history ORDER BY
start_time DESC;
```
    - Query for the maximum memory peak of the operator among all DNs, average memory peak among all DNs, and memory usage skew ratio across DNs. The memory unit is MB.  

```
SELECT plan_node_name,max_peak_memory,average_peak_memory,memory_skew_percent
FROM gs_wlm_operator_history ORDER BY start_time DESC;
```
    - Query for the maximum data spill volume of the operator among all DNs, average data spill volume among all DNs, and spill skew ratio across DNs. The data volume unit is MB.  

```
SELECT plan_node_name,max_spill_size,average_spill_size,spill_skew_percent FROM
gs_wlm_operator_history ORDER BY start_time DESC;
```
    - Query for the maximum execution time of the operator among all DNs, total execution time on all DNs, and execution time skew ratio across DNs. The time unit is ms.  

```
SELECT plan_node_name,max_cpu_time,total_cpu_time,cpu_skew_percent FROM
gs_wlm_operator_history ORDER BY start_time DESC;
```

## Precautions

- If queries are abnormally terminated due to **FATAL** or **PANIC** errors, their status will be displayed as **aborted** and no detailed information will be recorded.
- Status information in the optimization phase of query parsing cannot be monitored.
- Prefixes **gs** and **pgxc** indicate historical views showing single CN information and those showing cluster information, respectively. Common users can query only the views with the prefix **gs**.
- The number of data records that can be retained in memory is limited by the reserved memory size. After a real-time query is complete, its information is imported to a historical view and will be deleted from the history view 3 minutes after being stored in a hash table in memory. The upper limit of records varies by view. For a query-level view, whether a record can be stored depends on the upper limit of records allowed by the memory at that time point. On each CN, the maximum number of real-time query-level records

that can be stored in memory is **max\_session\_realt\_num** (set to **15240** by default). The maximum number of historical records that can be stored in memory is **max\_session\_hist\_num** (set to **18995** by default). The average execution time of queries in a service system is greater than **run\_time** (in the unit of s). The maximum number of concurrent jobs allowed by real-time views on each CN is equal to **num\_realt\_active**, and also to **max\_session\_realt\_num**. The maximum number of concurrent jobs allowed by historical views on each CN is equal to **num\_hist\_active**, and to **max\_session\_hist\_num/(180/run\_time)**.

- For operator-level views, whether a record can be stored depends on the upper limit allowed by the memory at that time point. If the number of plan nodes plus the number of records in the memory exceeds the upper limit, the record cannot be stored. On each CN, the maximum numbers of real-time and historical operator-level records that can be stored in memory are **max\_oper\_realt\_num** (set to **93622** by default) and **max\_oper\_hist\_num** (set to **187245** by default), respectively. The average number of plan nodes for queries is **num\_plan\_node**. The maximum number of concurrent jobs allowed by real-time views on each CN is equal to **num\_realt\_active**, and also to **max\_oper\_realt\_num/num\_plan\_node**. The maximum number of concurrent jobs allowed by historical views on each CN is equal to **num\_hist\_active**, and to **max\_oper\_hist\_num/(180/run\_time)/num\_plan\_node**.

## 14.6.6 Example for Querying for Top SQLs

In this section, TPC-DS sample data is used as an example to describe how to query for [Real-Time TopSQL](#) and [Historical TopSQL](#).

### Configuring Cluster Parameters

To query for historical or archived resource monitoring information about jobs of top SQLs, you need to set related GUC parameters first. The procedure is as follows:

1. Log in to the GaussDB(DWS) management console.
2. On the **Cluster Management** page, locate the required cluster and click the cluster name. The cluster details page is displayed.
3. Click the **Parameter Modifications** tab to view the values of cluster parameters.
4. Set an appropriate value for parameter [resource\\_track\\_duration](#) and click **Save**.

#### NOTE

If **enable\_resource\_record** is set to **on**, storage space expansion may occur and thereby slightly affects the performance. Therefore, set it to **off** if record archiving is unnecessary.

5. Go back to the **Cluster Management** page, click the refresh button in the upper right corner, and wait until the cluster parameter settings are applied.

### Example for Querying for Top SQLs

The TPC-DS sample data is used as an example.

- Step 1** Open the SQL client tool and connect to your database.
- Step 2** Run the **EXPLAIN** statement to query for the estimated cost of the SQL statement to be executed to determine whether resources of the SQL statement will be monitored.

By default, only resources of a query whose execution cost is greater than the value (default: **100000**) of **resource\_track\_cost** are monitored and can be queried by users.

For example, run the following statements to query for the estimated execution cost of the SQL statement:

```
SET CURRENT_SCHEMA = tpcds;
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 = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

In the following query result, the value in the first row of the **E-costs** column is the estimated cost of the SQL statement.

**Figure 14-7** EXPLAIN result

| id | operation                               | E-rows | E-width | E-costs |
|----|-----------------------------------------|--------|---------|---------|
| 1  | -> Row Adapter                          | 6      | 20      | 153.06  |
| 2  | -> Vector Limit                         | 6      | 20      | 153.06  |
| 3  | -> Vector Streaming (type: GATHER)      | 6      | 20      | 153.06  |
| 4  | -> Vector Limit                         | 6      | 20      | 152.84  |
| 5  | -> Vector Sort                          | 6      | 20      | 152.84  |
| 6  | -> Vector Hash Join (7,26)              | 6      | 20      | 152.83  |
| 7  | -> Vector Streaming(type: REDISTRIBUTE) | 6      | 4       | 134.57  |
| 8  | -> Vector Hash Join (9,18)              | 6      | 4       | 134.46  |
| 9  | -> Vector Hash Join (10,11)             | 1      | 44      | 97.33   |
| 10 | -> CStore Scan on store                 | 1      | 4       | 60.23   |
| 11 | -> Vector Subquery Scan on ctr1         | 6      | 40      | 37.07   |
| 12 | -> Vector Hash Aggregate                | 6      | 54      | 37.06   |
| 13 | -> Vector Streaming(type: REDISTRIBUTE) | 6      | 22      | 37.04   |
| 14 | -> Vector Hash Join (15,17)             | 6      | 22      | 37.00   |
| 15 | -> Vector Streaming(type: BROADCAST)    | 6      | 4       | 18.74   |
| 16 | -> CStore Scan on date_dim              | 1      | 4       | 18.06   |
| 17 | -> CStore Scan on store_returns         | 60     | 26      | 18.02   |
| 18 | -> Vector Hash Aggregate                | 6      | 68      | 37.09   |
| 19 | -> Vector Subquery Scan on ctr2         | 6      | 36      | 37.07   |
| 20 | -> Vector Hash Aggregate                | 6      | 54      | 37.06   |
| 21 | -> Vector Streaming(type: REDISTRIBUTE) | 6      | 22      | 37.04   |
| 22 | -> Vector Hash Join (23,25)             | 6      | 22      | 37.00   |
| 23 | -> Vector Streaming(type: BROADCAST)    | 6      | 4       | 18.74   |
| 24 | -> CStore Scan on date_dim              | 1      | 4       | 18.06   |
| 25 | -> CStore Scan on store_returns         | 60     | 26      | 18.02   |
| 26 | -> CStore Scan on customer              | 60     | 24      | 18.02   |

(26 rows)

In this example, to demonstrate the resource monitoring function of top SQLs, you need to set **resource\_track\_cost** to a value smaller than the estimated cost in the **EXPLAIN** result, for example, **100**. For details about the parameter setting, see [resource\\_track\\_cost](#).

 **NOTE**

After completing this example, you still need to reset **resource\_track\_cost** to its default value **100000** or a proper value. An overly small parameter value will compromise the database performance.

**Step 3** Run SQL statements.

```
SET CURRENT_SCHEMA = tpcds;
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 = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

**Step 4** During statement execution, query for the real-time memory peak information about the SQL statement on the current CN.

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM
gs_wlm_session_statistics ORDER BY start_time DESC;
```

The preceding command queries for the real-time peak information at the query-level. The peak information includes the maximum memory peak among all DNs per second, average memory peak among all DNs per second, and memory usage skew across DNs.

For more examples of querying for the real-time resource monitoring information of top SQLs, see [Typical Queries for Real-Time Top SQLs](#).

**Step 5** Wait until the SQL statement execution in [Step 3](#) is complete, and then query for the historical resource monitoring information of the statement.

```
select query,start_time,finish_time,duration,status from gs_wlm_session_history order by start_time desc;
```

The preceding command queries for the historical information at the query-level. The peak information includes the execution start time, execution duration (unit: ms), and execution status. The time unit is ms.

For more examples of querying for the historical resource monitoring information of top SQLs, see [Typical Queries for Historical Top SQLs](#).

**Step 6** Wait for 3 minutes after the execution of the SQL statement in [Step 3](#) is complete, query for the historical resource monitoring information of the statement in the **info** view.

If **enable\_resource\_record** is set to **on** and the execution time of the SQL statement in [Step 3](#) is no less than the value of **resource\_track\_duration**, historical information about the SQL statement will be archived to the **gs\_wlm\_session\_info** view 3 minutes after the execution of the SQL statement is complete.

The **info** view can be queried only when the postgres database is connected. Therefore, switch to the postgres database before running the following statement:

```
select query,start_time,finish_time,duration,status from gs_wlm_session_info order by start_time desc;
```

----**End**

# 15 PL/Java User-defined Functions

---

With the GaussDB(DWS) PL/Java functions, you can choose your favorite Java IDE to write Java methods and install the JAR files containing these methods into the GaussDB(DWS) database before invoking them. GaussDB(DWS) PL/Java is developed based on open-source Greenplum PL/Java 1.4.0. PL/Java uses JDK 1.8.0\_201.

## Constraints

Java UDF can be used for some Java logical computing. You are not advised to encapsulate services in Java UDF.

- You are not advised to connect to a database in any way (for example, JDBC) in Java functions.
- Currently, only data types listed in [Table 15-1](#) are supported. Other data types, such as user-defined data types and complex data types (for example, Java array and its derived types) are not supported.
- Currently, UDAF and UDTF are not supported.

## Examples

Before using PL/Java, you need to pack the implementation of Java methods into a JAR package and deploy it into the database. Then, create functions as a database administrator. For compatibility purposes, use JDK 1.8.0\_201 for compilation.

### Step 1 Compile a JAR package.

Java method implementation and JAR package archiving can be achieved in an integrated development environment (IDE). The following is a simple example of compilation and archiving through command lines. You can create a JAR package that contains a single method in the similar way.

First, prepare an **Example.java** file that contains a method for converting substrings to uppercase. In the following example, **Example** is the class name and **upperString** is the method name:

```
public class Example
{
 public static String upperString (String text, int beginIndex, int endIndex)
```

```
{
 return text.substring(beginIndex, endIndex).toUpperCase();
}
```

Then, create a **manifest.txt** file containing the following content:

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 08/14/2018 10:09 AM
```

**Manifest-Version** specifies the version of the **manifest** file. **Main-Class** specifies the main class used by the .jar file. **Specification-Title** and **Specification-Version** are the extended attributes of the package. **Specification-Title** specifies the title of the extended specification and **Specification-Version** specifies the version of the extended specification. **Created-By** specifies the person who created the file. **Build-Date** specifies the date when the file was created.

Finally, archive the .java file and package it into **javaudf-example.jar**.

```
javac Example.java
jar cfm javaudf-example.jar manifest.txt Example.class
```

#### NOTICE

JAR package names must comply with JDK rules. If a name contains invalid characters, an error occurs when a function is deployed or used.

### Step 2 Deploy the JAR package.

First store the JAR package on an OBS server. For details, see [Uploading an Object](#) in the *Object Storage Service Console Operation Guide*. Then, create the access key AK/SK. To obtain access keys, log in to the management console, move the cursor over the username in the upper right corner, and click **My Credentials**. Then choose **Access Keys** in the navigation tree on the left. On the **Access Keys** page, you can view the existing AKs or click **Create Access Key** to create an AK/SK pair. After that, log in to the database, run the `gs_extend_library` function, and import the package to GaussDB(DWS).

```
SELECT gs_extend_library('addjar', 'obs://bucket/path/javaudf-example.jar
accesskey=access_key_value_to_be_replaced secretkey=secret_access_key_value_to_be_replaced
region=region_name libraryname=example');
```

For details about how to use the `gs_extend_library` function, see [Manage JAR packages and files](#). Change the values of AK and SK as needed. Replace *region\_name* with an actual region name.

### Step 3 Use a PL/Java function.

Log in to the database as a user who has the **sysadmin** permission (for example, `dbadmin`) and create the `java_upperstring` function:

```
CREATE FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER)
 RETURNS VARCHAR
 AS 'Example.upperString'
LANGUAGE JAVA;
```

 NOTE

- The data type defined in the `java_upperstring` function should be a type in GaussDB(DWS) and match the data type defined in [Step 1](#) in the `upperString` method in Java. For details about the mapping between GaussDB(DWS) and Java data types, see [Table 15-1](#).
- The `AS` clause specifies the class name and static method name of the Java method invoked by the function. The format is *Class name.Method name*. The class name and method name must match the Java class and method defined in [Step 1](#).
- To use PL/Java functions, set **LANGUAGE** to **JAVA**.
- For details about `CREATE FUNCTION`, see [Create functions](#).

Execute the `java_upperstring` function.

```
SELECT java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
java_upperstring

T
(1 row)
```

**Step 4** Authorize a common user to use the PL/Java function.

Create a common user whose name is **udf\_user** and password is **GAUSS@123**.

```
CREATE USER udf_user PASSWORD 'GAUSS@123';
```

This command grants user **udf\_user** the permission for the `java_upperstring` function. Note that the user can use this function only if it also has the permission for using the schema of the function.

```
GRANT ALL PRIVILEGES ON SCHEMA public TO udf_user;
GRANT ALL PRIVILEGES ON FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER) TO udf_user;
```

Log in to the database as user **udf\_user**.

```
SET SESSION SESSION AUTHORIZATION udf_user PASSWORD 'GAUSS@123';
```

Execute the `java_upperstring` function.

```
SELECT public.java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
java_upperstring

T
(1 row)
```

**Step 5** Delete the function.

If you no longer need this function, delete it.

```
DROP FUNCTION java_upperstring;
```

**Step 6** Uninstall the JAR package.

Use the `gs_extend_library` function to uninstall the JAR package.

```
SELECT gs_extend_library('rmjar', 'libraryname=example');
```

----End



## SQL Definition and Usage

- **Manage JAR packages and files.**

A database user having the **sysadmin** permission can use the **gs\_extend\_library** function to deploy, view, and delete JAR packages in the database. The syntax of the function is as follows:

```
SELECT gs_extend_library('action', '[operation]');
```

### NOTE

- **action:** operation action. The options are as follows:
    - **ls:** Displays JAR packages in the database and checks the MD5 value consistency of files on each node.
    - **addjar:** deploys a JAR package on the OBS server in the database.
    - **rmjar:** Deletes JAR packages from the database.
  - **operation:** operation string. The format can be either of the following:  
 obs://[bucket]/[source\_filepath] accesskey=[accesskey] secretkey=[secretkey]  
 region=[region] libraryname=[libraryname]
    - **bucket:** name of the bucket to which the obs file belongs. It is mandatory.
    - **source\_filepath:** file path on the OBS server. Only .jar files are supported.
    - **accesskey:** key obtained for accessing the OBS service. It is mandatory.
    - **secret\_key:** secret key obtained for the OBS service. It is mandatory.
    - **region:** region where the OBS bucket stored in the JAR package of a user-defined function belongs to. This parameter is mandatory.
    - **libraryname:** user-defined library name, which is used to invoke JAR files in GaussDB(DWS). If **action** is set to **addjar** or **rmjar**, **libraryname** must be specified. If **action** is set to **ls**, **libraryname** is optional. Note that a user-defined library name cannot contain the following characters: /;&\$<>'\{}"() []~\*?!
  - Create functions.
- PL/Java functions can be created using the **CREATE FUNCTION** syntax and are defined as **LANGUAGE JAVA**, including the **RETURNS** and **AS** clauses.
- To use **CREATE FUNCTION**, specify the name and parameter type for the function to be created.
  - The **RETURNS** clause specifies the return type for the function.
  - The **AS** clause specifies the class name and static method name of the Java method to be invoked. If the **NULL** value needs to be transferred to the Java method as an input parameter, specify the name of the Java encapsulation class corresponding to the parameter type. For details, see [NULL Handling](#).
  - For details about the syntax, see **CREATE FUNCTION**.

```
CREATE [OR REPLACE] FUNCTION function_name
([{ argname [argmode] argtype [{ DEFAULT | := | = } expression] } [, ...]])
[RETURNS rettype [DETERMINISTIC]]
LANGUAGE JAVA
[
 { IMMUTABLE | STATBLE | VOLATILE }
 | [NOT] LEAKPROOF
 | WINDOW
 | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
 | [{ EXTERNAL } SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | AUTHID DEFINER |
 AUTHID CURRENT_USER }
 | { FENCED }
 | COST execution_cost
 | ROWS result_rows
```

```
| SET configuration_parameter { {TO |=} value | FROM CURRENT}
] [...]
{
 AS 'class_name.method_name' ({ argtype } [, ...])
}
```

- Use functions.

During execution, PL/Java searches for the Java class specified by a function among all the deployed JAR packages, which are ranked by name in alphabetical order, invokes the Java method in the first found class, and returns results.

- Delete functions.

PL/Java functions can be deleted by using the **DROP FUNCTION** syntax. For details about the syntax, see **DROP FUNCTION**.

```
DROP FUNCTION [IF EXISTS] function_name [(({ argmode } [argname] argtype } [, ...]))
[CASCADE | RESTRICT] ;
```

To delete an overloaded function (for details, see [Overloaded Functions](#)), specify **argtype** in the function. To delete other functions, simply specify **function\_name**.

- Authorize permissions for functions.

Only user **sysadmin** can create PL/Java functions. It can also grant other users the permission to use the PL/Java functions. For details about the syntax, see **GRANT**.

```
GRANT { EXECUTE | 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] ;
```

## Mapping for Basic Data Types

Table 15-1 PL/Java mapping for default data types

| GaussDB(DWS) | Java             |
|--------------|------------------|
| BOOLEAN      | boolean          |
| "char"       | byte             |
| bytea        | byte[]           |
| SMALLINT     | short            |
| INTEGER      | int              |
| BIGINT       | long             |
| FLOAT4       | float            |
| FLOAT8       | double           |
| CHAR         | java.lang.String |
| VARCHAR      | java.lang.String |
| TEXT         | java.lang.String |

| GaussDB(DWS) | Java                                               |
|--------------|----------------------------------------------------|
| name         | java.lang.String                                   |
| DATE         | java.sql.Timestamp                                 |
| TIME         | java.sql.Time (stored value treated as local time) |
| TIMETZ       | java.sql.Time                                      |
| TIMESTAMP    | java.sql.Timestamp                                 |
| TIMESTAMPTZ  | java.sql.Timestamp                                 |

## Array Type Processing

GaussDB(DWS) can convert basic array types. You only need to append a pair of square brackets ([]) to the data type when creating a function.

```
CREATE FUNCTION java_arrayLength(INTEGER[])
 RETURNS INTEGER
 AS 'Example.getArrayLength'
 LANGUAGE JAVA;
```

Java code is similar to the following:

```
public class Example
{
 public static int getArrayLength(Integer[] intArray)
 {
 return intArray.length;
 }
}
```

Invoke the following statement:

```
SELECT java_arrayLength(ARRAY[1, 2, 3]);
```

The expected result is as follows:

```
java_arrayLength

3
(1 row)
```

## NULL Handling

NULL values cannot be handled for GaussDB(DWS) data types that are mapped and can be converted to simple Java types by default. If you use a Java function to obtain and process the **NULL** value transferred from GaussDB(DWS), specify the Java encapsulation class in the **AS** clause as follows:

```
CREATE FUNCTION java_countnulls(INTEGER[])
 RETURNS INTEGER
 AS 'Example.countNulls(java.lang.Integer[])'
 LANGUAGE JAVA;
```

Java code is similar to the following:

```
public class Example
{
```

```
public static int countNulls(Integer[] intArray)
{
 int nullCount = 0;
 for (int idx = 0; idx < intArray.length; ++idx)
 {
 if (intArray[idx] == null)
 nullCount++;
 }
 return nullCount;
}
```

Invoke the following statement:

```
SELECT java_countNulls(ARRAY[null, 1, null, 2, null]);
```

The expected result is as follows:

```
java_countNulls

3
(1 row)
```

## Overloaded Functions

PL/Java supports overloaded functions. You can create functions with the same name or invoke overloaded functions from Java code. The procedure is as follows:

### Step 1 Create overloaded functions.

For example, create two Java methods with the same name, and specify the methods `dummy(int)` and `dummy(String)` with different parameter types.

```
public class Example
{
 public static int dummy(int value)
 {
 return value*2;
 }
 public static String dummy(String value)
 {
 return value;
 }
}
```

In addition, create two functions with the same names as the above two functions in GaussDB(DWS).

```
CREATE FUNCTION java_dummy(INTEGER)
 RETURNS INTEGER
 AS 'Example.dummy'
 LANGUAGE JAVA;

CREATE FUNCTION java_dummy(VARCHAR)
 RETURNS VARCHAR
 AS 'Example.dummy'
 LANGUAGE JAVA;
```

### Step 2 Invoke the overloaded functions.

GaussDB(DWS) invokes the functions that match the specified parameter type. The results of invoking the above two functions are as follows:

```
SELECT java_dummy(5);
java_dummy

10
```

```
(1 row)

SELECT java_dummy('5');
java_dummy

5
(1 row)
```

Note that GaussDB(DWS) may implicitly convert data types. Therefore, you are advised to specify the parameter type when invoking an overloaded function.

```
SELECT java_dummy(5::varchar);
java_dummy

5
(1 row)
```

In this case, the specified parameter type is preferentially used for matching. If there is no Java method matching the specified parameter type, the system implicitly converts the parameter and searches for Java methods based on the conversion result.

```
SELECT java_dummy(5::INTEGER);
java_dummy

10
(1 row)

DROP FUNCTION java_dummy(INTEGER);

SELECT java_dummy(5::INTEGER);
java_dummy

5
(1 row)
```

#### NOTICE

Data types supporting implicit conversion are as follows:

- **SMALLINT**: It can be converted to the **INTEGER** type by default.
- **SMALLINT** and **INTEGER**: They can be converted to the **BIGINT** type by default.
- **TINYINT**, **SMALLINT**, **INTEGER**, and **BIGINT**: They can be converted to the **BOOL** type by default.
- **CHAR**, **NAME**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **VARCHAR**, **NVARCHAR2**, **DATE**, **TIMESTAMP**, **TIMESTAMPZ**, **NUMERIC**, and **SMALLDATETIME**: They can be converted to the **TEXT** type by default.
- **TEXT**, **CHAR**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **DATE**, **NVARCHAR2**, **TIMESTAMP**, **NUMERIC**, and **SMALLDATETIME**: They can be converted to the **VARCHAR** type by default.

### Step 3 Delete the overloaded functions.

To delete an overloaded function, specify the parameter type for the function. Otherwise, the function cannot be deleted.

```
DROP FUNCTION java_dummy(INTEGER);
```

----End

## GUC Parameters

- **pljava\_vmoptions**

A session-level GUC parameter. It is used to set JVM startup parameters.

```
SET pljava_vmoptions='-Xmx64m -Xms2m -XX:MaxMetaspaceSize=8m';
```

**pljava\_vmoptions** supports:

- JDK8 JVM startup parameters.
- JDK8 JVM system attributes (starting with **-D**, for example, -**Djava.ext.dirs**).

---

### NOTICE

You are not advised to set any parameters that contain directories because such setting may lead to unpredictable behavior.

- User-defined parameters (starting with **-D**, for example, -**Duser.defined.option**).

---

### NOTICE

If a user sets **pljava\_vmoptions** to a value beyond the value range, an error will be reported during function revoking.

```
SET pljava_vmoptions=' illegal.option';
SET
```

```
SELECT java_dummy(5::int);
```

```
ERROR: UDF Error:cannot use PL/Java before successfully completing its setup.Please check if
your pljava_vmoption is set correctly,since we do not ignore illegal parameters.Or check the log
for more messages.
```

- **udf\_memory\_limit**

A system-level GUC parameter. It is used to limit the physical memory used by each CN or DN for executing UDFs. The default value is **200 MB**. You can use the **postgresql.conf** file to modify the parameter setting. The modification takes effect only after the database is restarted.

**NOTICE**

- **udf\_memory\_limit** is a part of **max\_process\_memory**. When a CN or DN is started, memory calculated by **udf\_memory\_limit** minus **200 MB** will be reserved for Worker processes. CN and DN processes are different from Worker processes, and the former will save memory for the latter.

For example, if **max\_process\_memory** is set to **10 GB** on a DN and **udf\_memory\_limit** is set to **4 GB**, the DN can use a maximum of 6.2 GB memory, that is, 10 GB – (4 GB – 200 MB). This case applies even if no UDF is executed. By default, the value of **udf\_memory\_limit** is **200 MB**. Querying the **pv\_total\_memory\_detail** view will prove that the value of **process\_used\_memory** would never exceed the calculation result of **max\_process\_memory – (udf\_memory\_limit – 200 MB)**.

- Executing a simplest Java UDF on a CN consumes about 50 MB physical memory. You can set this parameter based on the memory usage and concurrency of Java functions to be used. After this parameter is added, you are not advised to set **UDFWorkerMemHardLimit** and **FencedUDFMemoryLimit**.
- If the parallelism of the UDF process is excessively high and the memory usage exceeds the **udf\_memory\_limit** value, unexpected situations such as process exit may occur. In this scenario, the execution result may be unreliable. You are advised to set this parameter to reserve sufficient memory based on the site requirements. If the memory is severely insufficient, the UDF master process may exit. You can view the UDF master log for analysis. The default UDF master log path is **\$GAUSSLOG/cm/cm\_agent/pg\_log**. For example, if the following log is displayed, the memory resources are insufficient and the UDF master process exits. In this case, you need to check the **udf\_memory\_limit** parameter.

0 [BACKEND] FATAL: poll() failed: Bad address, please check the parameter:udf\_memory\_limit to make sure there is enough memory.

- **FencedUDFMemoryLimit**

A session-level GUC parameter. It is used to specify the maximum virtual memory used by a single Fenced UDF Worker process initiated by a session.

```
SET FencedUDFMemoryLimit='512MB';
```

The value range of this parameter is **(150 MB, 1 G]**. If the value is greater than **1 G**, an error will be reported immediately. If the value is less than or equals to **150 MB**, an error will be reported during function invoking.

**NOTICE**

- If **FencedUDFMemoryLimit** is set to **0**, the virtual memory for a Fenced UDF Worker process will not be limited.
- You are advised to use **udf\_memory\_limit** to control the physical memory used by Fenced UDF Worker processes. You are not advised to use **FencedUDFMemoryLimit**, especially when Java UDFs are used. If you are clear about the impact of this parameter, set it based on the following information:
  - After a C Fenced UDF Worker process is started, it will occupy about 200 MB virtual memory, and about 16 MB physical memory.
  - After a Java Fenced UDF Worker process is started, it will occupy about 2.5 GB virtual memory, and about 50 MB physical memory.

## Exception Handling

If there is an exception in a JVM, PL/Java will export JVM stack information during the exception to a client.

## Logging

PL/Java uses the standard Java Logger. Therefore, you can record logs as follows:

```
Logger.getAnonymousLogger().config("Time is " + new
Date(System.currentTimeMillis()));
```

An initialized Java Logger class is set to the **CONFIG** level by default, corresponding to the **LOG** level in GaussDB(DWS). In this case, log messages generated by Java Logger are all redirected to the GaussDB(DWS) backend. Then, the log messages are written into server logs or displayed on the user interface. MPPDB server logs record information at the **LOG**, **WARNING**, and **ERROR** levels. The SQL user interface displays logs at the **WARNING** and **ERROR** levels. The following table lists mapping between Java Logger levels and GaussDB(DWS) log levels.

**Table 15-2** PL/Java log levels

| java.util.logging.Level | GaussDB(DWS) Log Level |
|-------------------------|------------------------|
| <b>SERVER</b>           | ERROR                  |
| <b>WARINING</b>         | WARNING                |
| <b>CONFIG</b>           | LOG                    |
| <b>INFO</b>             | INFO                   |
| <b>FINE</b>             | DEBUG1                 |
| <b>FINER</b>            | DEBUG2                 |
| <b>FINEST</b>           | DEBUG3                 |



You can change Java Logger levels. For example, if the Java Logger level is changed to **SEVERE** by the following Java code, log messages (**msg**) will not be recorded in GaussDB(DWS) logs during **WARNING** logging.

```
Logger log = Logger.getAnonymousLogger();
Log.setLevel(Level.SEVERE);
log.log(Level.WARNING, msg);
```

## Security Issues

In GaussDB(DWS), PL/Java is an untrusted language. Only user **sysdamin** can create PL/Java functions. The user can grant other users the permission to use the PL/Java functions. For details, see [Authorize permissions for functions](#).

In addition, PL/Java controls user access to file systems, forbidding users from reading most system files, or writing, deleting, or executing any system files in Java methods.

# 16 Query Performance Optimization

---

## 16.1 Query Performance Tuning Overview

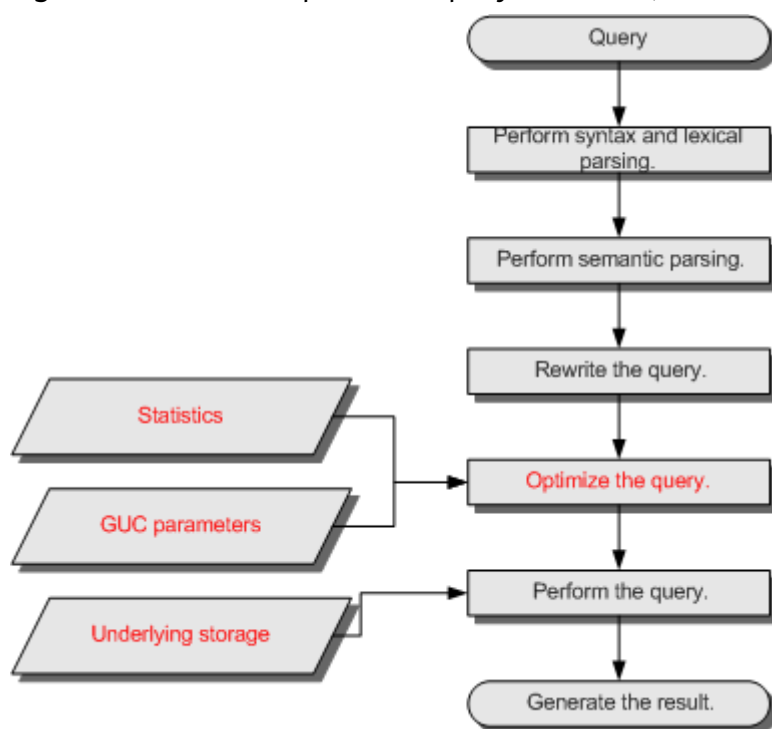
The aim of SQL tuning is to maximize the utilization of resources, including CPU, memory, disk I/O, and network I/O. To maximize resource utilization is to run SQL statements as efficiently as possible to achieve the highest performance at a lower cost. For example, when performing a typical point query, you can use the seqscan and filter (that is, read every tuple and query conditions for match). You can also use an index scan, which can be implemented at a lower cost but achieve the same effect.

This chapter describes how to analyze and improve query performance, and provides common cases and troubleshooting methods.

## 16.2 Analyzing Queries

### 16.2.1 Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in [Figure 16-1](#) and [Table 16-1](#). The texts in red are steps where database administrators can optimize queries.

**Figure 16-1** Execution process of query-related SQL statements by the SQL engine**Table 16-1** Execution process of query-related SQL statements by the SQL engine

| Procedure                              | 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 last 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 last step and the internal database statistics. For details about how the internal database statistics and GUC parameters affect the query optimization (execution plan), see <a href="#">Optimizing Queries Using Statistics</a> and <a href="#">Optimizing Queries Using GUC parameters</a> . |
| 5. Perform the query.                  | Executes the SQL statements based on the execution path specified in the last step. Selecting a proper underlying storage mode improves the query execution efficiency. For details, see <a href="#">Optimizing Queries Using the Underlying Storage</a> .                                                                                                                                                 |

## Optimizing Queries Using Statistics

The GaussDB(DWS) optimizer is a typical Cost-based Optimization (CBO). By using CBO, the database calculates the number of tuples and the execution cost for each execution step under each execution plan based on the number of table tuples, column width, NULL record ratio, and characteristic values, such as distinct, MCV, and HB values, and certain cost calculation methods. The database then selects the execution plan that takes the lowest cost for the overall execution or for the return of the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the 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.

## Optimizing Queries Using GUC parameters

Optimizing queries aims to select an efficient execution mode.

Take the following 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(DWS) supports nested loop, merge join, and hash join. The optimizer estimates the result set value and the execution cost under each join mode based on the statistics of the **customer** and **store\_sales** tables and selects the execution plan that takes the lowest execution cost.

As described in the preceding content, 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 the GUC parameters.

## Optimizing Queries Using the Underlying Storage

GaussDB(DWS) supports row- and column-store tables. The selection of an underlying storage mode strongly depends on specific customer business scenarios. You are advised to use column-store tables for computing service scenarios (mainly involving association and aggregation operations) and row-store tables for service scenarios, such as point queries and massive **UPDATE** or **DELETE** executions.

Optimization methods of each storage mode will be described in details in the performance optimization chapter.

## 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 practical experience.

This requires that the system administrators know the customer business well and have professional knowledge of SQL statements.

## 16.2.2 SQL Execution Plan

The SQL execution plan is a node tree, which displays detailed procedure when GaussDB(DWS) 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** provides one row for each execution node, displaying the basic node type and the cost estimation that the optimizer made for the execution of this node, as shown in [Figure 16-2](#).

**Figure 16-2** 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 Joining 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 scanning 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 possible way to perform these operations, so different execution node types may be displayed here.
- The first row (the upper-layer node) estimates the total execution cost of the execution plan. This value indicates the value that the optimizer tries to minimize.

## Execution Plan Display Format

GaussDB(DWS) provides four display formats: **normal**, **pretty**, **summary**, and **run**.

- **normal** indicates that the default printing format is used in [Figure 16-2](#).
- **pretty** indicates that the optimized display mode of GaussDB(DWS) is used. The new format contains a plan node ID, effectively analyzing performance, as shown in [Figure 16-3](#).
- **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 16-3** Example of an execution plan using the pretty format

```
postgres=# 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 details. The following lists the common **EXPLAIN** syntax. For details, 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 **PERFORMANCE** on a query sometimes takes longer time than executing the query normally. 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.

The **EXPLAIN PERFORMANCE** lightweight execution is consistent with **EXPLAIN PERFORMANCE** but greatly reduces the time spent on performance analysis.

### 16.2.3 SQL Execution Plan Details

As described in [SQL Execution Plan](#), **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. In this section, detailed execution plan and execution information are described.

## Execution Plan

The following SQL statement is used as an example:

```
select
 cxjh,
 count(1)
from dwcjk
group by cxjh;
```

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

```
postgres=# explain select cxjh, count(1) from dwcjk group by cxjh;
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)
```

#### Interpretation of the execution plan column (horizontal):

- **id**: execution operator node ID
- **operation**: execution node operator name

The operator of the Vector prefix refers to a vectorized execution engine operator, which exists in a query containing a column-store table.

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 are defined by the optimizer based on cost parameters, habitually grasping disk page as a unit. Other overhead parameters are set by referring to E-costs.
  - The cost of each node (the E-costs value) includes the cost of all of its child nodes.
  - Overhead reflects only what the optimizer is concerned about, but does not consider the time that the result row passed 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: **CStore Scan on dwcjk**

The table scan operator scans the table **dwcjk** using Cstore Scan. The function of this layer is to read data in the table dwcjk from the buffer or disks, or transfers it to the upper-layer node to participate in the calculation.

2. Layer 2: **Vector Hash Aggregate**

Aggregation operators are used to perform aggregation operations (group by) on operators calculated from the lower layer.

3. Layer 3: **Vector Streaming (type: GATHER)**

The GATHER-typed Shuffle operator aggregates data from DN to the CN.

4. Layer 4: **Row Adapter**

Storage format conversion operator is used to convert data in columns of the memory to data in rows for client display.

It should be noted that when operators in the top layer are Data Node Scan, you need to set **enable\_fast\_query\_shipping** to **off** to view detailed execution plan as follows:

```
explain select cjxh, count(1) from dwcjk group by cjxh;
QUERY PLAN

Data Node Scan (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes
(2 rows)
```

After **enable\_fast\_query\_shipping** is set, the execution plan is displayed as follows:

```
postgres=# set enable_fast_query_shipping=off;
SET
postgres=# 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)
```

### 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 first sort the location of index identifier rows based on physical locations before reading them. This minimizes the independent capturing overhead.

If there are separate indexes on multiple columns referenced in **WHERE**, the optimizer might choose to use an **AND** or **OR** combination of the indexes. However, this requires the visiting of both indexes, so it is not



necessarily a win compared to using just one index and treating the other condition as a filter.

The following Index scans featured with different sorting mechanisms are involved:

- **Bitmap Index Scan**  
Fetches data pages using a bitmap.
- **Index Scan using index\_name**  
Fetches table rows in index order, which makes them even more expensive to read. However, there are so few rows that the extra cost of sorting the row locations is unnecessary. This plan type is used mainly for queries fetching just a single row and queries having an **ORDER BY** condition that matches the index order, because no extra sorting step is needed to satisfy **ORDER BY**.

## 2. Table connection modes

### – Nested Loop

Nested-loop is used for queries that have a smaller data set connected. In a Nested-loop join, the foreign table drives the internal table and each row returned from the foreign table should have a matching row in the internal table. The returned result set of all queries should be less than 10,000. The table that returns a smaller subset will work as a foreign table, and indexes are recommended for connection fields of the internal 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 10000 rows. As a result, the cost is not decreased. It increases a bit (by 10000 x **cpu\_operator\_cost**) to reflect the extra CPU time spent on checking the **WHERE** condition.

### – LIMIT

**LIMIT** limits the number of output execution results. If a **LIMIT** condition is added, not all rows are retrieved.

## Task Execution

You can use **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** to check the SQL statement execution information and compare the actual execution and the optimizer's estimation to find what to optimize. **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:

```
postgres=# 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)
```

This figure shows that the execution information can be classified into the following 7 aspects.

- 
- 429

- **A-time:** indicates the execution completion time of the current operator. Generally, the A-time of the operator executed on the DN is two values enclosed by square brackets ([]), indicating the shortest time and longest time for completing the operator on all DNs, respectively.
  - **A-rows:** number of actual output tuples of the operator
  - **E-distinct:** the estimated distinct value of the hashjoin operator
  - **Peak Memory:** peak memory of the operator on each DN
  - **A-width:** the current operator tuple actual width of each line. This parameter is valid only for the heavy memory operator is displayed, including: (Vec)HashJoin, (Vec)HashAgg, (Vec) HashSetOp, (Vec)Sort, and (Vec)Materialize operator. The (Vec)HashJoin calculation of width is the width of the right subtree operator, it will be displayed in the right subtree.
2. Predicate Information (identified by plan id):  
This part displays the static information that does not change during 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**, split 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 in 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, initialization, execution, and time in the end phase on each CN, and the system available memory during the current statement execution, and statement estimation memory information.

**NOTICE**

- The difference between **A-rows** and **E-rows** shows the deviation between the optimizer estimation and actual execution. Generally, if the deviation is large, the plan generated by the optimizer is untrusted, and you need to modify the deviation value.
- If the difference of the A-time values is large, it indicates that the operator computing skew (difference between execution time on DN) is large and that manual performance tuning is 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 memory parameter during SQL statement optimization. Generally, the output from **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** is provided for the input for further optimization.
- In **Query Summary**, the value of **Total runtime** is greater than the value of **A-time** of the operator whose ID is 1. **A-time** indicates the time for executing SQL statements only. **Total runtime** includes the process of collecting execution information of all instances and processing the execution information based on the output format. If the cluster scale is large, the execution plan is complex and the deviation between the two values is large.

## 16.2.4 Querying SQL Statements That Affect Performance Most

This section describes how to query SQL statements whose execution takes a long time, leading to poor system performance.

### Procedure

**Step 1** Run the following command to connect to the database:

```
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 A 8.0.0 build af002019) compiled at 2020-01-10 05:43:20 commit 6995 last mr 11566)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

**Step 2** Query the statements that are run for a long time in the database.

```
SELECT current_timestamp - query_start AS runtime, datname, username, query FROM pg_stat_activity
where state != 'idle' ORDER BY 1 desc;
```

After the query, query statements are returned as a list, ranked by execution time in descending order. The first result is the query statement that has the longest execution time in the system. The returned result contains the SQL statement invoked by the system and the SQL statement run by users. Find the statements that were run by users and took a long time.

Alternatively, you can set **current\_timestamp - query\_start** to be greater than a threshold to identify query statements that are executed for a duration longer than this threshold.

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval '1 days';
```

**Step 3** 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 4** View the running query statements.

Viewing **pg\_stat\_activity** is used as an example here.

```
SELECT datname, username, state FROM pg_stat_activity;
datname | username | state |
-----+-----+-----+
postgres | omm | idle |
postgres | omm | active |
(2 rows)
```

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

To identify only active query statements, run the following command:

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

**Step 5** Analyze the status of the query statements that were run for a long time.

- If the query statement is normal, wait until the execution is complete.
- If a query statement is blocked, run the following command to view this query statement:

```
SELECT datname, username, state, query FROM pg_stat_activity WHERE waiting = true;
```

The command output lists a query statement in the block state. The lock resource requested by this query statement is occupied by another session, so this query statement is waiting for the session to release the lock resource.

 **NOTE**

Only when the query is blocked by internal lock resources, the **waiting** field is **true**. In most cases, block happens when query statements are waiting for lock resources to be released. However, query statements may be blocked because they are waiting to write in files or for timers. Such blocked queries are not displayed in the **pg\_stat\_activity** view.

----End

## 16.2.5 Checking Blocked Statements

During database running, query statements are blocked in some service scenarios and run for an excessively long time. In this case, you can forcibly terminate the faulty session.

### Procedure

**Step 1** Run the following command to connect to the database:

```
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 A 8.0.0 build af002019) compiled at 2020-01-10 05:43:20 commit 6995 last mr 11566)
Non-SSL connection (SSL connection is recommended when requiring high-security)
```

```
Type "help" for help.
```

```
postgres=#
```

- Step 2** View blocked query statements and information about the tables and schemas that block the query statements.

```
SELECT w.query as waiting_query,
 w.pid as w_pid,
 w.username as w_user,
 l.query as locking_query,
 l.pid as l_pid,
 l.username as l_user,
 t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid
where w.waiting;
```

The thread ID, user information, query status, as well as information about the tables and schemas that block the query statements are returned.

- Step 3** Run the following command to terminate the required session, where **139834762094352** is the thread ID:

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

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

```
PG_TERMINATE_BACKEND

t
(1 row)
```

If a command output similar to the following is displayed, a user is attempting to terminate the session, and the session will be reconnected rather than being terminated.

```
FATAL: terminating connection due to administrator command
FATAL: terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

#### NOTE

If the **PG\_TERMINATE\_BACKEND** function is used to terminate the background threads of the session, the **gsql** client will be reconnected rather than be logged out.

----End

## 16.3 Tuning Queries

### 16.3.1 Tuning 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 statistics are unavailable or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance

problem occurred because no statistics are collected. For details, see [Updating Statistics](#).

- Step 2** View the execution plan to find out the cause. If the SQL statements have been running for a long period of time and not ended, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to check the execution plan and actual running situation and then accurately locate the fault. For details about the execution plan, see [SQL Execution Plan](#).
- Step 3** [Reviewing and Modifying 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. The query rewriting method is universal in all databases.
- Step 6** Specify a join order; join, stream, or scan operations; number of rows in a result; or redistribution skew information to tune an execution plan, improving query performance. For details, see [Hint-based Tuning](#).
- Step 7** Routinely [maintain tables](#) and [rebuild indexes](#).
- Step 8** (Optional) Improve performance by using operators if resources are sufficient in GaussDB(DWS). For details, see [Recommended Suggestions for SMP](#).

----End

## 16.3.2 Updating Statistics

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.

### Context

The **ANALYZE** statement collects statistic about table contents in databases, which will be stored in the system table **PG\_STATISTIC**. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertion 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.

If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan.



## Procedure

Run the following commands to update the statistics about a table or the entire database:

```
ANALYZE tablename; --Update statistics about a table.
ANALYZE; ---Update statistics about the entire database.
```

Run the following statements to perform statistics-related operations on multiple columns:

```
ANALYZE tablename ((column_1, column_2)); --Collect statistics about column_1 and
column_2 of tablename.

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2)); --Declare statistics about column_1
and column_2 of tablename.
ANALYZE tablename; --Collect statistics about one or more columns.

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --Delete statistics about column_1
and column_2 of tablename or their statistics declaration.
```

### NOTICE

After the statistics are declared for multiple columns by running the **ALTER TABLE *tablename* ADD STATISTICS** statement, the system collects the statistics about these columns next time **ANALYZE** is performed on the table or the entire database.

To collect the statistics, run the **ANALYZE** statement.

### NOTE

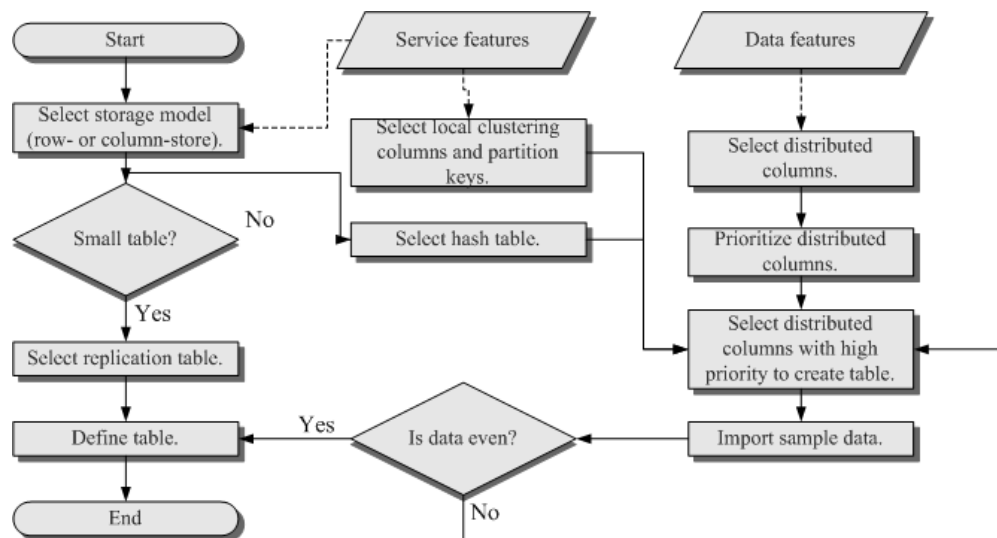
Use **EXPLAIN** to show the execution plan of each SQL statement. If **rows=10** (the default value, probably indicating the table has not been analyzed) is displayed in the **SEQ SCAN** output of a table, run the **ANALYZE** statement for this table.

## 16.3.3 Reviewing and Modifying a Table Definition

In a distributed framework, data is distributed on DNs. Data on one or more DNs is stored on a physical storage device. To properly define a table, you must:

1. **Evenly distribute data on each DN** to avoid the available capacity decrease of a cluster caused by insufficient storage space of the storage device associated with a DN. Specifically, select a proper distribution key to avoid data skew.
2. **Evenly assign table scanning tasks on each DN** to avoid that a 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. **Avoid the use of random I/O** by using clustering or partial clustering.
5. **Avoid data shuffle** to reduce the network pressure by selecting join condition columns or **GROUP BY** columns as the distribution key.

The distribution key is the core for defining a table. [Figure 16-4](#) 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 16-4** Procedure of defining a table

For details about how to review and modify table definitions, see [Learning the Tutorial: Tuning Table Design](#).

## 16.3.4 Typical SQL Optimization Methods

### 16.3.4.1 Typical SQL Optimization Methods

SQL tuning involves continuous analysis and adjustment. You need to test-run a query, locate and fix its performance issues (if any) based on its execution plan, and run it again, until the execution performance meet your requirements. This chapter describes common methods of SQL tuning.

### 16.3.4.2 Optimizing SQL Self-Diagnosis

Performance issues may occur when you query data or run the **INSERT**, **DELETE**, **UPDATE**, or **CREATE TABLE AS** statement. In this case, you can query the **warning** column in the [gs\\_wlm\\_session\\_statistics](#), [gs\\_wlm\\_session\\_history](#), and [gs\\_wlm\\_session\\_info](#) views to obtain reference for performance optimization.

Alarms that can trigger SQL self diagnosis depend on the settings of [resource\\_track\\_level](#). If **resource\_track\_level** is set to **query**, alarms about the failures in collecting column statistics and pushing down SQL statements will trigger the diagnosis. If **resource\_track\_level** is set to **operator**, all alarms will trigger the diagnosis.

Whether a SQL plan will be diagnosed depends on the settings of [resource\\_track\\_cost](#). A SQL plan will be diagnosed only if its execution cost is greater than **resource\_track\_cost**. You can use the **EXPLAIN** keyword to check the plan execution cost.

## Alarms

Currently, the following performance alarms will be reported:

- Some column statistics are not collected.

An alarm will be reported if some column statistics are not collected. For details about the optimization, see [Updating Statistics](#) and [Optimizing Statistics](#).

An alarm will also be reported if column statistics are not collected for queries on OBS or HDFS foreign tables. The performance of the **ANALYZE** statement executed for these foreign tables is poor and seems inefficient for a simple query. Run this statement as needed.

Example alarms:

No statistics about a table are not collected.

```
Statistic Not Collect:
schema_test.t1
```

The statistics about a single column are not collected.

```
Statistic Not Collect:
schema_test.t2(c1,c2)
```

The statistics about multiple columns are not collected.

```
Statistic Not Collect:
schema_test.t3((c1,c2))
```

The statistics about a single column and multiple columns are not collected.

```
Statistic Not Collect:
schema_test.t4(c1,c2) schema_test.t4((c1,c2))
```

- SQL statements are not pushed down.

The cause details are displayed in the alarms. For details about the optimization, see [Optimizing Statement Pushdown](#).

- If the pushdown failure is caused by functions, the function names are displayed in the alarm.
- If the pushdown failure is caused by syntax, the alarm indicates that the syntax does not support pushdown. For example, the syntax containing the **With Recursive**, **Distinct On**, or **Row** expression, and syntax whose return value is of record type do not support pushdown.

Example alarms:

```
SQL is not plan-shipping, reason : ""enable_stream_operator" is off"
SQL is not plan-shipping, reason : ""Distinct On" can not be shipped"
SQL is not plan-shipping, reason : ""v_test_unshipping_log" is VIEW that will be treated as Record type can't be shipped"
```

- In a hash join, the larger table is used as the inner table.

An alarm will be reported if the number of rows in the inner table reaches or exceeds 10 times of that in the outer table, more than 100 thousand of inner-table rows are processed on each DN in average, and the join statement has spilled to disk. You can view the **query\_plan** column in [gs\\_wlm\\_session\\_history](#) to check whether the hash join is used. For details about the optimization, see [Hint-based Tuning](#).

Example alarm:

```
PlanNode[7] Large Table is INNER in HashJoin "Vector Hash Aggregate"
```

- nestloop** is used in a large-table equivalent join.

An alarm will be reported if **nestloop** is used in an equivalent join where more than 100 thousand of the larger-table rows are processed on each DN in average. You can view the **query\_plan** field of [gs\\_wlm\\_session\\_history](#) to check whether **nestloop** is used. For details about the optimization, see [Hint-based Tuning](#).

Example alarm:

```
PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

- A large table is broadcasted.

An alarm will be reported if more than 100 thousand of rows are broadcasted on each DN in average. For details about the optimization, see [Hint-based Tuning](#).

Example alarm:

```
PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

- Data skew occurs.

An alarm will be reported if the number of rows processed on any DN exceeds 100 thousand, and the number of rows processed on a DN reaches or exceeds 10 times of that processed on another DN. For the optimization, see [Case: Selecting an Appropriate Distribution Column](#) and [Optimizing Data Skew](#).

Example alarm:

```
PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

- The index is improper.

During base table scanning, an alarm is reported if the following conditions are met:

- For row-store tables:
  - When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/1000 and the number of output lines is greater than 10,000.
  - When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/1000, the number of output lines is less than or equal to 10,000, and the number of scanned lines is greater than 10,000.
- For column-store tables:
  - When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/10000 and the number of output lines is greater than 100.
  - When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/10,000, the number of output lines is less than or equal to 100, and the number of scanned lines is greater than 10,000.

For details about the optimization, see [Optimizing Operators](#).

Example alarms:

```
PlanNode[4] Indexscan is not properly used:"Index Only Scan", output:524288, filtered:0, rate:1.00000
PlanNode[5] Indexscan is ought to be used:"Seq Scan", output:1, filtered:524288, rate:0.00000
```

- Estimation is inaccurate.

An alarm will be reported if the maximum number or the estimated maximum number of rows processed on a DN is over 10 thousand, and the larger number reaches or exceeds 10 times of the smaller one. For details about the optimization, see [Hint-based Tuning](#).

Example alarm:

```
PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

## Restrictions

1. An alarm contains a maximum of 2048 characters. If the length of an alarm exceeds this value (for example, a large number of long table names and column names are displayed in the alarm when their statistics are not collected), a warning instead of an alarm will be reported.  
WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
2. If a query statement contains the **Limit** operator, alarms of operators lower than **Limit** will not be reported.
3. For alarms about data skew and inaccurate estimation, only alarms on the lower-layer nodes in a plan tree will be reported. This is because the same alarms on the upper-level nodes may be triggered by problems on the lower-layer nodes. For example, if data skew occurs on the **Scan** node, data skew may also occur in operators (for example, **Hashagg**) at the upper layer.

### 16.3.4.3 Optimizing Statement Pushdown

#### Statement Pushdown

Currently, the GaussDB(DWS) optimizer can use three methods to develop statement execution policies in the distributed framework: generating a statement pushdown plan, a distributed execution plan, or a distributed execution plan for sending statements.

- A statement pushdown plan pushes query statements from a CN down to DNs for execution and returns the execution results to the CN.
- In a distributed execution plan, a CN compiles and optimizes query statements, generates a plan tree, and then sends the plan tree to DNs for execution. After the statements have been executed, execution results will be returned to the CN.
- A distributed execution plan for sending statements pushes queries that can be pushed down (mostly base table scanning statements) to DNs for execution. Then, the plan obtains the intermediate results and sends them to the CN, on which the remaining queries are to be executed.

The third policy sends many intermediate results from the DNs to a 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 not advised to use the query statements that only the third policy is applicable to.

Statements cannot be pushed down to DNs 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.

## Viewing 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, the SQL statements cannot be pushed down to DNs. If the execution plan contains Streaming, the SQL statements can be pushed down to DNs.

For example:

```
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:

```
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);
CREATE TABLE test_stream(a int, b float);--float does not support redistribution.
CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- The **returning** statement cannot be pushed down.

```
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)
- Aggregate functions using the **ORDER BY** statement cannot be pushed down.
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 columns in **count(distinct expr)** do not support redistribution, they do not support pushdown.
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)
 - Statements using **distinct on** cannot be pushed down.
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)
 - In a statement using **FULL JOIN**, if the column specified using **JOIN** does not support redistribution, the statement does not support pushdown.
explain select * from test_stream t1 full join test_stream t2 on t1.a=t2.b;

QUERY PLAN
----- Hash Full Join (cost=0.38..0.82 rows=30 width=24)
Hash Cond: ((t1.a)::double precision = t2.b)
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=12)
Node/s: All datanodes
-> Hash (cost=0.00..0.00 rows=30 width=12)
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=12)
Node/s: All datanodes
(7 rows)
 - Does not support array expression pushdown.
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 or HDFS tables.	LOG: SQL can't be shipped, reason: RecursiveUnion contains HDFS Table or ForeignScan is not shippable (In this table, LOG describes the cause of not supporting pushdown.) In the current version, queries containing foreign tables or HDFS tables do not support pushdown.
2	Multiple Node Groups	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup 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	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;	LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches ALL is not used for UNION . In this case, the return result is deduplicated.
4	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;	LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable A base table contains the system catalog.

No.	Scenario	Cause of Not Supporting Pushdown
5	<pre>WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;</pre>	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable Only VALUES is used for scanning base tables. In this case, the statement can be executed on the CN, and DNs are unnecessary.
6	<pre>select a.ID,a.Name, (with recursive cte as (select ID, PID, NAME from b where b.ID = 1 union all select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id where child.ID = a.ID) select NAME from cte limit 1) cName from (select id, name, count(*) as cnt from a group by id,name) a order by 1,2;</pre>	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable The correlation conditions of correlated subqueries are only in the recursion part, and the non-recursion part has no correlation condition.
7	<pre>WITH recursive t_result AS (select * from(SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm < 10 order by dm limit 6 offset 2) UNION all SELECT t2.dm,t2.sj_dm,t2.name ' > ' t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;</pre>	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) 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.

No.	Scenario	Cause of Not Supporting Pushdown
8	<pre>with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;</pre>	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" recursive of multiple-layers are nested. That is, a recursive is nested in the recursion part of another recursive.

Functions That Do Not Support Pushdown

This module describes the variability of functions. The function variability in GaussDB(DWS) is as follows:

- **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 values, but that its result varies by SQL statements.

- **VOLATILE**

Indicates that the function value can change even within a single table scan, so no optimizations can be made.

The volatility of a function can be obtained by querying 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 are not pushed down, because pushdown may lead to incorrect results.

For a UDF, you can specify the values of **provolatile** and **proshippable** during its creation. 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 UDF function, check whether its **provolatile** and **proshippable** are correctly defined.

Example: UDF

Define a user-defined function that generates fixed output for a certain input as the **immutable** type.

Take the sales information of TPCDS 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 returns certain output when certain input is entered. 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)
```

func_percent_1 is pushed down to DNs for quicker execution. (In TPCDS 1000X, where three CNs and 18 DNs are used, the query efficiency is improved by over 100 times).

Example 2: Pushing Down the Sorting Operation

For details, see [Case: Pushing Down Sort Operations to DNs](#).

16.3.4.4 Optimizing Subqueries

What Is a Subquery

A **SELECT...FROM...WHERE...** statement is called a query block. Nesting a query block in the **FROM**, **WHERE**, or **HAVING** clause of another query is called a nested query. Queries nested in other queries are nested subqueries or subqueries for short.

Subqueries can be categorized into correlated subqueries and uncorrelated subqueries. Execution of a correlated subquery depends on some attributes of its parent query and the parameters delivered by the parent query. If some of the parameters have been changed, you need to execute the subquery again to obtain a new result. Uncorrelated subqueries are executed separately, and you need to only execute each once.

Optimization Example

Uncorrelated subqueries do not affect the query performance, because each of them is executed once only. Correlated subqueries need to be executed more than once, greatly affecting the query performance. Two common examples are as follows:

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 and create an index on the **a** column.

2. Modify the **SELECT** statement, change the subquery to a **JOIN** relationship between the primary table and the parent query, or modify the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
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:

```
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.

16.3.4.5 Optimizing Statistics

What Is Statistic Optimization

GaussDB(DWS) 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 in 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 NULL3
, 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
) with (orientation = column, COMPRESSION = MIDDLE) 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  
with (orientation = column, COMPRESSION = MIDDLE) 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

For details, see [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 checked in an Internet cafe within 15 minutes before and after the check-in of a specified person.

```
SELECT  
C.WBM,  
C.DZQH,  
C.DZ,
```

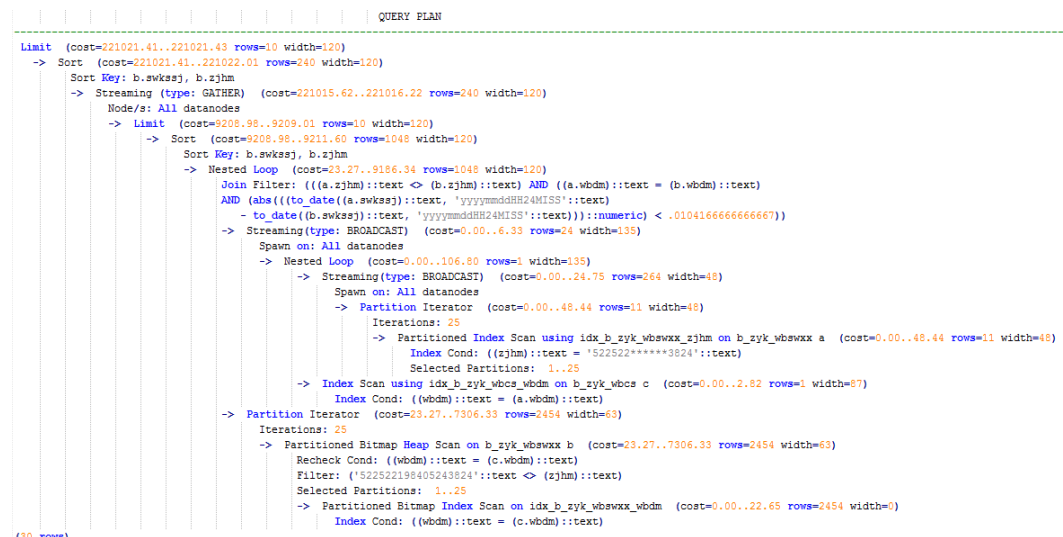
```

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 16-5 shows the execution plan. This query takes about 12s.

Figure 16-5 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 IN** 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)

```

```

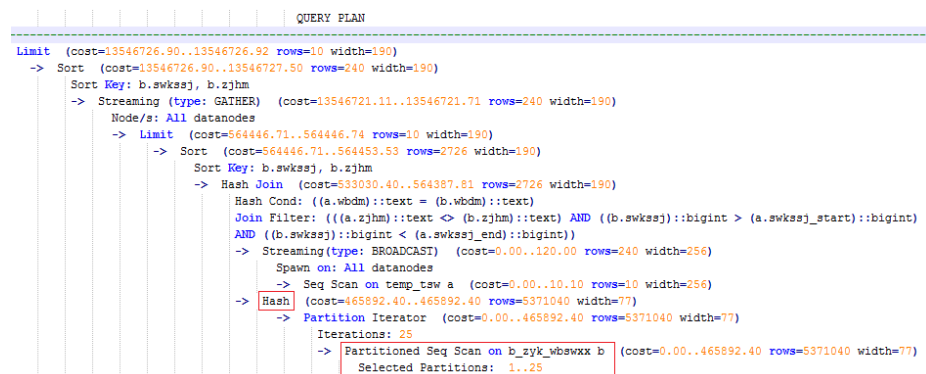
)
;
//Insert the Internet access record of the specified person, and process the start time and end time.
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
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 check in an Internet cafe before and after 15 minutes of the check-in
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 16-6](#) shows the execution plan.

Figure 16-6 Using an unlogged table (2)



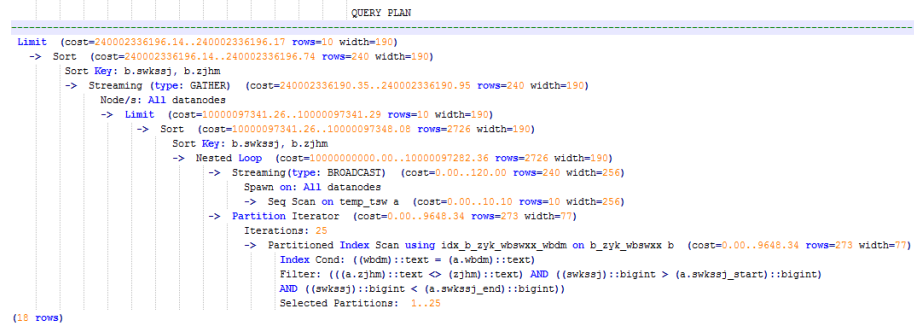
- In the previous plan, **Hash Join** has been executed, and a Hash table has been created for the large table **b_zyk_wbswxx**. The table contains large amounts of data, so the creation takes long time.

temp_tsw contains only hundreds of records, and an equal-value connection is created between **temp_tsw** and **b_zyk_wbswxx** using wbdm (the 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.

- Execute the following statement to change **JOIN** to **NEST LOOP JOIN**.
SET enable_hashjoin = off;

Figure 16-7 shows the execution plan. The query takes about 3s.

Figure 16-7 Using an unlogged table (3)



- 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, you are advised to use the unlogged table 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.swksj_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 global statistics using **ANALYZE** improves query performance.

If a performance problem occurs, you can use plan hint to adjust the query plan to the previous one. For details, see [Hint-based Tuning](#).

16.3.4.6 Optimizing Operators

What Is Operator Optimization

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/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	100000000	237060	19KB			20	20933222.75
2	-> Vector Streaming (type: GATHER)	55664.220	100000000	237060	241KB			20	20933222.75
3	-> Vector Hash Aggregate	[55124.685, 55132.180]	100000000	237060	[29349KB, 29441KB]	16MB	[20, 20]	20	20918406.50
4	-> Vector Streaming (type: REDISTRIBUTE)	[55119.781, 55129.779]	339364604	4856184	[1219KB, 1219KB]	1MB		20	10461210.85
5	-> Vector Hash Aggregate	[35875.656, 51016.424]	339364604	4856184	[732850KB, 746894KB]	16MB	[20, 20]	20	10457195.65
6	-> Vector Partition Iterator	[9035.202, 13565.884]	970000000	935838097	[9KB, 9KB]	1MB		20	10195891.68
7	-> Partitioned Cost Scan on xujie_mp_day_energy_dev_1	[9015.645, 13535.346]	970000000	935838097	[845KB, 845KB]	1MB		20	10195891.68

(7 rows)

Operator Optimization 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.

```
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
create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
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=off** is set to disable merge join and **enable_nestloop=off** is set to disable **NestLoop** so that the optimizer selects hash join, the join takes more than 200 ms.

```
postgres=# 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 | 101KB | | | | 0 | 48629179.77
3 | -> Vector Streaming (type: GATHER) | 184300.186 | 4 | 4 | 181KB | | | | 0 | 48629179.77
4 | -> Vector Aggregate         | [165575.384,184252.368] | 2880404 | 2880404 | [1490KB, 1490KB] | 1MB | | | 0 | 48629179.61
5 | -> Vector Hash Loop (6,7) | [162918.849,121438.162] | 2880404 | 2880404 | [748B, 748B] | 1MB | | | 0 | 48627379.35
6 | -> CStore Scan on store_sales ss | [15.460,16.229] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | [8,8] | | 4 | 16683.10
7 | -> Vector Materialize       | [118314.521,132478.454] | 12968211302 | 18000 | [869KB, 900KB] | 10MB | | | 4 | 3890.00
8 | -> CStore Scan on item i | [0.234,0.243] | 18000 | 18000 | [476KB, 476KB] | 1MB | | | 4 | 3867.50
(8 rows)
```

```
postgres=# set enable_nestloop=off;
SET
postgres=# set enable_mergejoin=off;
SET
postgres=# explain analyze select count(*) fpostgres=# ales 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 | 188KB | | | | 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 | [0.234,0.246] | 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**.

```
postgres=# 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     | [1784.800,1883.243] | 18000 | 17644 | [273KB, 273KB] | 1MB | | | 4 | 92186.02
4 | -> Vector Sort              | [1752.270,1845.357] | 2880404 | 2880404 | [12846KB, 135135KB] | 16MB | [8,8] | | 4 | 98541.40
5 | -> CStore Scan on store_sales | [12.483,13.548] | 2880404 | 2880404 | [490KB, 490KB] | 1MB | | | 4 | 16683.10
(5 rows)
```

```
postgres=# set enable_sort=off;
SET
postgres=# 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              | 838.218 | 18000 | 17644 | 20KB | | | | 4 | 21016.93
2 | -> Vector Streaming (type: GATHER) | 834.264 | 18000 | 17644 | 228KB | | | | 4 | 21016.93
3 | -> Vector Hash Aggregate     | [753.017,758.204] | 18000 | 17644 | [26252KB, 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)
```

16.3.4.7 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(DWS) provides a complete solution for data skew, including storage and computing skew.

Data Skew in the Storage Layer

In the GaussDB(DWS) 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 column is not properly selected. This can be solved by adjusting the distribution column.

For example:

```
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, you can view the number of scan rows of each DN in the inventory table. The number of rows of each DN differs a lot, the biggest is 63000000 and the smallest value is 15000000. 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 column. Run `table_skewness()` to view data skew of each DN in the inventory table. The query result is as follows:

```
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 column, which causes a data skew. Based on the data distribution of each column, change the distribution column to **inv_item_sk**. The skew status is as follows:

```
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 current **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 columns. 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 columns 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 DN's and causes a skew.

```
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 **skew_option** parameter. 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 columns 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 filtering conditions, or after the join of other tables, we cannot determine whether the skewed data still exists 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 provide the skew information, based on which the optimizer optimize 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_REDISIBUTE_PART_ROUNDROBIN** on the side with skew. Specifically, perform round-robin on skew data and redistribution on non-skew data.

Use **PART_REDISIBUTE_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_REDISIBUTE_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_REDISIBUTE_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 --	Vector 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 is

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**) is balanced and the end-to-end performance of the query is doubled.

– Aggregate optimization

For aggregation, data on each DN is deduplicated based on the **GROUP BY** key and then redistributed. After the deduplication on DNs, 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)

Applicable scope

- **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.

16.3.5 Adjusting Key Parameters During SQL Tuning

This section introduces key CN parameters that affect optimization of SQL statements in GaussDB(DWS). For details about the parameter configuration method, see the **Database Parameter** section.

Table 16-2 CN parameters

Parameter/ Reference Value	Description
enable_nestloop=off	Specifies how the optimizer uses Nest Loop Join. If this parameter is set to on, the optimizer preferentially uses Nest Loop Join. If it is set to off, the optimizer preferentially uses other methods, if any. NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), run the following SQL statement: <pre>SET enable_nestloop to on;</pre> By default, this parameter is set to off. Change the value as required. Generally, if the internal table of the Nest Loop can use indexes, you can set the parameter to on.

Parameter/ Reference Value	Description
enable_bitmapscan= on	Specifies whether the optimizer uses bitmap scanning. If the value is on, bitmap scanning is used. If the value is off, it is not used. NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), run the following SQL statement: SET enable_bitmapscan to off; The bitmap scanning applies only in the query condition where a > 1 and b > 1 and indexes are created on columns a and b. During performance tuning, if the query performance is poor and bitmapscan operators are in the execution plan, set this parameter to off and check whether the performance is improved.
enable_fast_query_ shipping=on	Specifies whether the optimizer uses a distribution framework. 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 CNs and then sent to DNs for execution. NOTE If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), run the following SQL statement: SET enable_fast_query_shipping to off;
enable_hashagg=o n	Specifies whether to enable the optimizer's use of Hash-aggregation plan types.
enable_hashjoin=o n	Specifies whether to enable the optimizer's use of Hash-join plan types.
enable_mergejoin= on	Specifies whether to enable the optimizer's use of Hash-merge plan types.
enable_indexscan= on	Specifies whether to enable the optimizer's use of index-scan plan types.
enable_indexonlysc an=on	Specifies whether to enable the optimizer's use of index-only-scan plan types.
enable_seqscan=on	Specifies whether the optimizer uses bitmap scanning. It is impossible to suppress sequential scans entirely, but setting this variable to off allows the optimizer to preferentially choose other methods if available.
enable_sort=on	Specifies the optimizer sorts. It is impossible to fully suppress explicit sorts, but setting this variable to off allows the optimizer to preferentially choose other methods if available.

Parameter/ Reference Value	Description
enable_broadcast=on	Specifies whether enable the optimizer's use of 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 LAZY_AGG and MAGIC_SET rewriting rules.

16.3.6 Hint-based Tuning

16.3.6.1 Plan Hint Optimization

In plan hints, you can specify a join order; join, stream, and scan operations; you can specify the number of rows in a result and redistribution skew information to tune an execution plan, improving query performance.

Function

The hint syntax must follow immediately after a **SELECT** keyword and is written in the following format:

```
/*+ <plan hint>*/
```

You can specify multiple hints for a query plan and separate them by 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.

For 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.

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.

Supported Hints

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 **tablescan**, **indexscan**, and **indexonlyscan**
- Sublink name hints
- Skew hints, supporting only the skew in the redistribution involving Join or HashAgg

Precautions

- Hints do not support **Agg**, **Sort**, **Setop**, or **Subplan**.
- Hints do not support SMP or Node Groups.

Examples

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_warehouse_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
;
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	6		273	3401632.49
2	-> Vector Streaming (type: GATHER)	6		273	3401632.49
3	-> Vector Hash Aggregate	6	16MB	273	3401630.82
4	-> Vector Streaming(type: REDISTRIBUTE)	6	1MB	169	3401630.78
5	-> Vector Hash Join (6,21)	6	16MB	169	3401630.42
6	-> Vector Hash Join (7,20)	7	43MB	173	3400343.15
7	-> Vector Streaming(type: REDISTRIBUTE)	7	1MB	123	3395775.64
8	-> Vector Hash Join (9,19)	7	27MB	123	3395775.48
9	-> Vector Streaming(type: REDISTRIBUTE)	7	1MB	123	3386294.72
10	-> Vector Hash Join (11,18)	7	16MB	123	3386294.56
11	-> Vector Hash Join (12,14)	7	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 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)

16.3.6.2 Join Order Hints

Function

These hints specify 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))
```

Parameter Description

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.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ leading((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store
store_sales))*/ i_product_name product_name ...
```

First, **store_sales** and **store** are joined and **store_sales** is the inner table. Then, The result is joined with **promotion**, **item**, **customer**, **ad2**, and **store_returns** in sequence. The optimized plan is as follows:

WARNING: Duplicated or conflict hint: Leading(store_sales store), will be discarded.

id	operation	E-rows	E-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](#).

16.3.6.3 Join Operation Hints

Function

Specifies the join method. It can be nested loop join, hash join, or merge join.

Syntax

```
[no] nestloop[hashjoin|mergejoin](table_list)
```

Parameter Description

- 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**.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

nestloop is used for the last join between **store_sales**, **store_returns**, and **item**. The optimized plan is as follows:

id	operation	E-rows	E-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)					

16.3.6.4 Rows Hints

Function

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)
```

Parameter Description

- #, +, -, 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, because this hint will take effect for a join as long as the two tables appear on both sides of this 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.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

Multiply the result set of joined **store_sales** and **store_returns** by 50. The optimized plan is as follows:

id	operation	E-rows	E-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.

16.3.6.5 Stream Operation Hints

Function

These hints specify a stream operation, which can be **broadcast** or **redistribute**.

Syntax

```
[no] broadcast|redistribute(table_list)
```

Parameter Description

- **no** indicates that the specified hint will not be used for a join.
- *table_list* specifies the tables to be joined. For details, see [Parameter Description](#).

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item
store) customer)) */ i_product_name product_name ...
```

In the original plan, the join result of **store_sales**, **store_returns**, **item**, and **store** is redistributed before it is joined with **customer**. After the hinting, the redistribution is disabled and the join order is retained. The optimized plan is as follows:

id	operation	E-rows	E-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)

16.3.6.6 Scan Operation Hints

Function

These hints specify a scan operation, which can be **tablescan**, **indexscan**, or **indexonlyscan**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

Parameter Description

- **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, column-store tables, HDFS tables, HDFS foreign tables, OBS tables, and subquery tables. HDFS tables include primary tables and delta tables. The delta tables are invisible to users. Therefore, scan operation hints are used only for primary tables.

Examples

To specify an index-based hint for a scan, create an index named **i** on the **i_item_sk** column of the **item** table.

```
create index i on item(i_item_sk);
```

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

item is scanned based on an index. The optimized plan is as follows:

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)					

16.3.6.7 Sublink Name Hints

Function

These hints specify the name of a sublink block.

Syntax

```
blockname (table)
```

Parameter Description

- *table* indicates the name you have specified for a sublink block.

NOTE

- This hint is used by an outer query only when a sublink is 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)

16.3.6.8 Skew Hints

Function

These hints specify 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)])

Parameter Description

- **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 contains 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. The format is **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.

Examples

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	1000006.10
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	294300.51
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	294300.51
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	34485.50
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 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;
```

16.3.6.9 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

- 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.

For 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 if a sublink contains multiple tables to be joined, because the table list in the sublink becomes invalid after the sublink is pulled up.
- Unsupported column types.
 - Skew hints are specified to optimize redistribution. They will be invalid if their corresponding columns do not support redistribution.
- Specified 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.

16.3.6.10 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 1000X+24DN environments. 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

In this plan, the performance of the layer-10 **broadcast** is poor because the estimation result generated at layer 11 is 2140 rows, which is 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 correlation 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]	2879997999	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]	60000000	60000000
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]	60000000	60000000
9	-> CStore Scan on public.customer_address	[69.877,121.460]	60000000	60000000
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

Data skew occurs after the join of **item** and **customer_address** because **item** is broadcasted 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

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	[3.150,3.530]	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
  );
```

16.3.7 Routinely Maintaining Tables

To ensure proper database running, after insert and deletion operations, you need to routinely run **VACUUM FULL** and **ANALYZE** based on customer scenarios and update statistics to obtain better performance.

Related Concepts

You need to routinely run **VACUUM**, **VACUUM FULL**, and **ANALYZE** to maintain tables, because:

- **VACUUM FULL** reclaims disk space occupied by updated or deleted data and combines small-size data files.
- **ANALYZE** collects statistics on tables in databases. The statistics are stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

Procedure

Step 1 Run the **VACUUM** or **VACUUM FULL** command to reclaim disk space.

- **VACUUM FULL:**
VACUUM FULL *customer*;
VACUUM

VACUUM FULL needs to add exclusive locks on tables it operates on and requires that all other database operations be suspended.

Step 2 Run **ANALYZE** to update statistics.

```
ANALYZE customer;  
ANALYZE
```

Run **ANALYZE VERBOSE** to update statistics and display table information.

```
ANALYZE VERBOSE customer;  
ANALYZE
```

You can use **VACUUM ANALYZE** at the same time to optimize the query.

```
VACUUM ANALYZE customer;  
VACUUM
```

NOTE

VACUUM and **ANALYZE** cause a substantial increase in I/O traffic, which may cause poor performance of other active sessions.

----End

Maintenance Suggestion

- Routinely run **VACUUM FULL** on large tables. If the database performance deteriorates, run **VACUUM FULL** on all tables in the database. If the database performance is stable, you are advised to monthly run **VACUUM FULL**.
- Routinely run **VACUUM FULL** on system catalogs, especially **PG_ATTRIBUTE**.
- The automatic cleanup process (**AUTOVACUUM**) in the system automatically runs the **VACUUM** and **ANALYZE** commands to recycle the record space in deleted status and update statistics in the table.

16.3.8 Routinely Recreating an Index

Context

When data deletion is repeatedly performed in the database, index keys will be deleted from the index page, resulting in index distention. Recreating an index routinely improves query efficiency.

The database supports B-tree, GIN, and PSORT.

- Recreating a B-tree index helps improve query efficiency.
 - If massive data is deleted, index keys on the index page will be deleted. As a result, the number of index pages reduces and index bloat occurs. Recreating an index helps reclaim wasted space.
 - In the created index, pages adjacent in its logical structure are adjacent in its physical structure. Therefore, a created index achieves higher access speed than an index that has been updated for multiple times.
- You are advised not to recreate a non-B-tree index.

Rebuilding an Index

Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete an index and run the **CREATE INDEX** statement to create an index.

When you delete an index, a temporary exclusive lock is added in the parent table to block related read/write operations. When you create an index, the write operation is locked but the read operation is not. The data is read and scanned by order.

- Run the **REINDEX** statement to recreate an index:
 - When you run the **REINDEX TABLE** statement to recreate an index, an exclusive lock is added to block related read/write operations.
 - When you run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table (), an exclusive lock is added to block read/write operations on the table.

Procedure

Assume the ordinary index `areaS_idx` exists in the **area_id** column of the imported table **areaS**. Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete the index and run the **CREATE INDEX** statement to create an index.
 - a. Delete an index.

```
DROP INDEX areaS_idx;  
DROP INDEX
```
 - b. Create an index.

```
CREATE INDEX areaS_idx ON areaS (area_id);  
CREATE INDEX
```
- Run the **REINDEX** statement to recreate an index.
 - Run the **REINDEX TABLE** statement to recreate an index.

```
REINDEX TABLE areaS;  
REINDEX
```
 - Run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table ().

```
REINDEX INTERNAL TABLE areaS;  
REINDEX
```

16.3.9 Configuration SMP

16.3.9.1 SMP Application Scenarios and Restrictions

The SMP feature improves the performance through operator parallelism and occupies more system resources, including CPU, memory, network, and I/O. Actually, SMP is a method consuming resources to save time. It improves system performance in appropriate scenarios and when resources are sufficient, but may deteriorate performance otherwise. SMP adaption is added to the 1.3.x version. With this feature, the optimal parallelism degree is dynamically selected based on resource types and query characteristics. The impacts of different resources on SMP performance are described as follows:

Application Scenarios

- Operators supporting parallel processing are used.

The execution plan contains the following operators that allow concurrent operations:

 - a. Scan: Row-store and column-store ordinary and partitioned tables, and ORC-formatted OBS foreign tables can be sequentially scanned. Foreign tables imported using GDS can be scanned in parallel. All of the above does not support replication tables.

- b. Join: HashJoin, NestLoop
- c. Agg: HashAgg, SortAgg, PlainAgg, and WindowAgg, which supports only **partition by**, and does not support **order by**.
- d. Stream: Redistribute, Broadcast
- e. Other: Result, Subquerscan, Unique, Material, Setop, Append, VectorRow, RowToVec
- SMP-unique operators

To execute queries in parallel, Stream operators are added for data exchange of the SMP feature. These new operators can be considered as the subtypes of Stream operators.

 - a. Local Gather aggregates data of parallel threads within a DN
 - b. Local Redistribute redistributes data based on the distributed key across threads within a DN
 - c. Local Broadcast broadcasts data to each thread within a DN.
 - d. Local RoundRobin distributes data in polling mode across threads within a DN.
 - e. Split Redistribute redistributes data across parallel threads on different DNs.
 - f. Split Broadcast broadcasts data to all parallel DN threads in the cluster.

Among these operators, Local operators exchange data between parallel threads within a DN, and non-Local operators exchange data across DNs.
- Example

The TPCB Q1 parallel plan is used as an example.

Figure 16-8 TPCB Q1 parallel execution plan

```

id | operation
-----+-----
1 | -> Row Adapter
2 | -> Vector Streaming (type: GATHER)
3 | -> Vector Sort
4 | -> Vector Streaming(type: LOCAL GATHER dop: 1/2)
5 | -> Vector Hash Aggregate
6 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)
7 | -> Vector Hash Aggregate
8 | -> Vector Partition Iterator
9 | -> Partitioned CStore Scan on public.lineitem
(9 rows)

```

In this plan, the Partitioned CStore Scan and HashAgg operators are processed in parallel, and the Local Gather and Split Redistribute operators are added.

In this example, No. 6 operator is Split Redistribute, and **dop: 2/2** next to the operator indicates that the degree of parallelism (DOP) of the sender and receiver is 2. No. 4 operator is Local Gather and is marked with **dop: 1/2**. The DOP of its sender thread is 2 and that of its receiver thread is 1. That is, No. 5 operator Hash Aggregate at a lower layer is executed with a DOP of 2, while the No. 1 to 3 operators at upper layers are serially executed. In this way, No. 4 operator aggregates parallel thread data within a DN.

You can view the parallelism situation of each operator in the dop information.

Non-applicable Scenarios

1. Operators are processed on CNs.
2. Statements that cannot be pushed down are executed.
3. The **subplan** and **initplan** of a query and operators containing a subquery are executed.

16.3.9.2 Resource Impact on SMP Performance

The SMP architecture uses abundant resources to obtain time. After the plan parallelism is executed, the resource consumption is added, including the CPU, memory, I/O, and network bandwidth resources. As the parallelism degree is expanded, the resource consumption increases. If these resources become a bottleneck, the SMP cannot improve the performance and the overall cluster performance may be deteriorated. The following information describes the situations that the SMP affects these resources:

- **CPU resources**

In a general customer scenario, the system CPU usage rate is not high. Using the SMP parallelism architecture will fully use the CPU resource to improve the system performance. If the number of CPU kernels of the database server is too small and the CPU usage is already high, enabling the SMP parallelism may deteriorate the system performance due to resource compete between multiple threads.

- **Memory resources**

After queries are performed concurrently, the memory usage increases. However, the upper limit of the memory usage of each operator is limited by the resource management module. If the system memory is insufficient, data may be saved to disks after queries are concurrently performed. As a result, the query performance deteriorates.

- **Network bandwidth resources**

To execute a query in parallel, data exchange operators are added. Local Stream operators exchange data between threads within a DN. Data is exchanged in memory and network performance is not affected. Non-Local operators exchange data over the network and increase network load. If the capacity of a network resource becomes a bottleneck, parallelism may also increase the network load.

- **I/O resources**

A parallel scan increases I/O resource consumption. It can improve performance only when I/O resources are sufficient.

16.3.9.3 Other Factors Affecting SMP Performance

Besides resource factors, there are other factors that impact the SMP parallelism performance, such as unevenly data distributed in a partitioned table and system parallelism degree.

- **Impact of data skew on SMP performance**

Serious data skew deteriorates parallel execution performance. For example, if the data volume of a value in the join column is much more than that of other values, the data volume of a parallel thread will be much more than

that of others after Hash-based data redistribution, resulting in the long-tail issue and poor parallelism performance.

- **Impact on the SMP performance due to system parallelism degree**

The SMP feature uses more resources, and unused resources are decreasing in a high concurrency scenario. Therefore, enabling the SMP parallelism will result in serious resource compete among queries. Once resource competes occur, no matter the CPU, I/O, memory, or network resources, all of them will result in entire performance deterioration. In the high concurrency scenario, enabling the SMP will not improve the performance effect and even may cause performance deterioration.

16.3.9.4 Recommended Suggestions for SMP

The summary is as follows based on the preceding analysis:

Restrictions

You need to meet the following conditions if you want to use the SMP feature to improve the performance:

The CPU, memory, I/O and network bandwidth resources are sufficient. The SMP system is a solution that uses abundant resources to exchange time. After the plan parallelism is executed, resource consumption is increased. When these resources become a bottleneck, the SMP cannot improve the performance and even the performance may be deteriorated. In the case of a resource bottleneck, you are advised to disable the SMP.

Procedure

1. Set **query_dop** to **1** (default value). Use **explain** to generate an execution plan and check whether the plan meets the restrictions in [SMP Application Scenarios and Restrictions](#). If the plan meets the restrictions, go to the next step.
2. Set **query_dop** to **0** (adaptive). The system dynamically selects the optimal parallelism degree [1,8] for each query based on the resource usage and query plans to improve the query performance as much as possible.
3. Set **query_dop=-value**. The value range of the parallelism degree is [1,value].
4. Set **query_dop=value**. The parallelism degree is 1 or value.

16.4 Optimization Cases

16.4.1 Case: Selecting an Appropriate Distribution Column

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 column 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, as shown in **Figure 16-9**.

Figure 16-9 Selecting an appropriate distribution column (1)

```
postgres=> 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 column of **t1** and **b** is the distribution column 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 16-10**.

Figure 16-10 Selecting an appropriate distribution column (2)

```
postgres=> 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)
```

16.4.2 Case: Creating an Appropriate Index

Symptom

Query the information about all personnel in the sales department.

```
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	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	69.801	34	2	[212KB]			354	53.67
2	-> Sort	[21.508,23.283]	34	2	[30KB, 36KB]	16MB	[380,380]	354	53.32
3	-> Hash Join (4,5)	[21.483,23.153]	34	2	[7KB, 7KB]	1MB		354	53.31
4	-> Seq Scan on hr.states	[0.007,0.022]	17	20	[12KB, 12KB]	1MB		110	13.13
5	-> Hash	[21.026,22.663]	34	2	[262KB, 294KB]	16MB	[284,284]	268	40.08
6	-> Streaming (type: REDISTRIBUTE)	[21.024,22.458]	34	2	[85KB, 85KB]	1MB		268	40.08
7	-> Hash Join (8,9)	[13.814,14.827]	34	2	[8KB, 8KB]	1MB		268	39.80
8	-> Seq Scan on hr.staffs	[0.035,0.043]	107	20	[19KB, 19KB]	1MB		190	13.13
9	-> Hash	[13.361,14.348]	2	4	[292KB, 292KB]	16MB	[124,124]	102	26.57
10	-> Streaming (type: BROADCAST)	[13.291,14.279]	2	4	[85KB, 85KB]	1MB		102	26.57
11	-> Hash Join (12,13)	[6.359,7.446]	1	2	[6KB, 6KB]	1MB		102	26.48
12	-> Seq Scan on hr.places	[0.008,0.018]	15	20	[14KB, 14KB]	1MB		102	13.13
13	-> Hash	[5.999,7.077]	1	1	[259KB, 291KB]	16MB	[32,32]	24	13.28
14	-> Streaming (type: REDISTRIBUTE)	[5.999,6.959]	1	1	[84KB, 85KB]	1MB		24	13.28
15	-> Seq Scan on hr.sections	[0.021,0.022]	1	1	[14KB, 14KB]	1MB		24	13.16
(15 rows)									
Predicate Information (identified by plan id)									
3 --Hash Join (4,5)									
Hash Cond: (states.state_id = places.state_id)									
7 --Hash Join (8,9)									
Hash Cond: (staffs.section_id = sections.section_id)									
11 --Hash Join (12,13)									
Hash Cond: (places.place_id = sections.place_id)									
15 --Seq Scan on hr.sections									
Filter: ((sections.section_name)::text = 'Sales':::text)									
Rows Removed by Filter: 26									
(9 rows)									

The optimized execution plan is as follows (two indexes have been created on the **places.place_id** and **states.state_id** columns):

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	50.411	34	2	[212KB]			354	46.84
2	-> Sort	[20.889,22.621]	34	2	[30KB, 36KB]	16MB	[380,380]	354	46.49
3	-> Nested Loop (4,13)	[20.870,22.490]	34	2	[59B, 59B]	1MB		354	46.48
4	-> Streaming (type: REDISTRIBUTE)	[20.869,21.852]	34	2	[85KB, 85KB]	1MB		268	35.46
5	-> Nested Loop (6,7)	[13.433,14.452]	34	2	[6KB, 6KB]	1MB		268	35.18
6	-> Seq Scan on hr.staffs	[0.027,0.032]	107	20	[19KB, 19KB]	1MB		190	13.13
7	-> Materialize	[13.237,14.230]	109	4	[10KB, 10KB]	16MB	[124,124]	102	21.66
8	-> Streaming (type: BROADCAST)	[13.150,14.137]	2	4	[85KB, 85KB]	1MB		102	21.65
9	-> Nested Loop (10,12)	[5.784,6.822]	1	2	[38B, 38B]	1MB		102	21.56
10	-> Streaming (type: REDISTRIBUTE)	[5.782,6.675]	1	1	[84KB, 85KB]	1MB		24	13.28
11	-> Seq Scan on hr.sections	[0.019,0.020]	1	1	[14KB, 14KB]	1MB		24	13.16
12	-> Index Scan using loc_id_pk on hr.places	[0.091,0.091]	1	2	[24KB, 24KB]	1MB		102	8.27
13	-> Index Scan using state_c_id_pk on hr.states	[0.352,0.352]	34	2	[23KB, 23KB]	1MB		110	5.50
(13 rows)									
Predicate Information (identified by plan id)									
5 --Nested Loop (6,7)									
Join Filter: (sections.section_id = staffs.section_id)									
Rows Removed by Join Filter: 73									
11 --Seq Scan on hr.sections									
Filter: ((sections.section_name)::text = 'Sales':::text)									
Rows Removed by Filter: 26									
12 --Index Scan using loc_id_pk on hr.places									
Index Cond: (places.place_id = sections.place_id)									
13 --Index Scan using state_c_id_pk on hr.states									
Index Cond: (states.state_id = places.state_id)									
(10 rows)									

16.4.3 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 16-11 shows the execution plan.

Figure 16-11 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	0.000.792	1	72	72KB			160	0.04244120.99	
2	→ Vector Streaming (type: GATHER)	0.000.779	1	72	144KB			160	0.04244120.99	
3	→ Vector Hash Aggregate	0.021.920,3679.684	1	1	1004KB, 9000KB	140B	[75, 78]	55	0.064807.23	
4	→ Vector Streaming (type: REDISTRIBUTE)	0.021.909,3679.593	72	2	2404KB, 2400KB	1MB		55	0.064807.23	
5	→ Vector Hash Aggregate	0.016.687,3634.515	72	2	1013KB, 501KB	140B	[75, 78]	55	0.064807.23	
6	→ Vector Hash Join (T, T)	0.036.674,3545.293	3665920	0.094077	17776KB, 5141KB	140B		55	0.064807.23	
7	→ Vector Hash Aggregate	0.000.673	1	1087848	1010KB, 1010KB	140B		52	0.071.95	
8	→ CScore Scan on dim_loc_bscrnc	0.071.1.229	1	1087848	1510B	1412KB, 1412KB	1MB	[48, 48]	32	0.071.95
9	→ Vector Hash Join (T, T)	0.041.130,3639.410	26319416	0.087825	12107KB, 2937KB	140B	[80, 80]	60	0.064807.23	
10	→ CScore Scan on sdr_web_bscrnc_1day_sdr	0.051.201,2751.915	142544506	0.223163	1311KB, 1311KB	1MB		64	0.051224.25	
11	→ CScore Scan on dim_rat_mapping rat	0.070.0.111	1	288	577KB, 577KB	1MB	[16, 16]	8	0.051.95	
(11 rows)										

Optimization Analysis

- As shown in Figure 16-11, 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 16-12 shows the execution plan.

Figure 16-12 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)	873.784	1	72	44KB				140 121433605.45
3	Vector Hash Aggregate	[685.940, 744.454]	1	1	[3004KB, 3005KB]	1MB			55 1686577.84
4	Vector Streaming (type: REDDISTRIBUTE)	[685.910, 744.365]	72	1	[2424KB, 2483KB]	1MB			55 1686577.84
5	Vector Hash Aggregate	[590.319, 710.912]	72	1	[3015KB, 3015KB]	1MB			55 1686577.84
6	Vector Hash Join (7,10)	[561.469, 661.631]	3663920	102203	[2769KB, 2769KB]	1MB			55 1684533.77
7	Vector Hash Join (7,9)	[545.846, 636.404]	366400	44859	[2330KB, 2330KB]	1MB			40 1594757.26
8	CStore Scan on sdr_web_bscrnc_id sdr	[441.494, 628.405]	366400	78509	[515KB, 535KB]	1MB			64 1595024.20
9	CStore Scan on dim_rat_mapping rat	[0.001, 0.107]	288	4	[577KB, 577KB]	1MB			4 150.03
10	Vector Subquery Scan on dim	[5.526, 6.960]	1087848	15109	[40KB, 40KB]	1MB			7 1726.04
11	Vector Hash Aggregate	[5.497, 6.931]	1087848	15109	[2539KB, 2539KB]	1MB			32 1574.95
12	CStore Scan on dim_loo_bscrnc	[1.087, 1.424]	1087848	15109	[1412KB, 1412KB]	1MB			32 1272.77

16.4.4 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 sorting 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_VOLUM_KPIID
FROM public.test AS test
GROUP BY IMSI
order by TOTAL_VOLUM_KPIID DESC;

```

The execution plan is as follows:

```

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 DN, as shown below:

```
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.

16.4.5 Setting cost_param and Optimizing Query Performance

Symptom 1

If **bit0** is set to **1** (**set cost_param=1**), an improved mechanism is used for estimating the selection rate of non-equi-joins. This method is more accurate for estimating the selection rate of joins between two identical tables. The following example describes the optimization scenario when **bit0** of **cost_param** is set to **1**. In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized formula is selected for calculation.

Note: The selection rate 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
) with (orientation = column, COMPRESSION = MIDDLE) 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
)with (orientation = column, COMPRESSION = MIDDLE) 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 **cost_param & bit0** is **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 **cost_param & bit0** 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** is set to **1** (**set cost_param=2**), 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. The following example describes the optimization scenario when **cost_param** & **bit1** is **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 **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 criteria and **lineitem.l_partkey** in the second filter criteria are two columns with strong relationship of the lineitem table. In this situation, when you estimate the rate of the hash join conditions, if **cost_param & bit1** is **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. The plan after optimization is shown as follows:

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)					

16.4.6 Case: Adjusting the Distribution Key

Symptom

During a site test, the information is displayed after **EXPLAIN ANALYZE** is executed:

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Streaming (type: GATHER)	[94158, 404]	0	670912	282KB	1MB	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, 586]	310	670912	[78B, 78B]	1MB	73	102533776.78	
5	HashAggregate	[93212, 425, 93419, 980]	310	670912	[145KB, 197KB]	10MB	45	102533446.74	
6	Streaming (type: REDISTRIBUTE)	[93212, 374, 93419, 924]	5886	670934	[2091KB, 2093KB]	1MB	45	102533305.05	
7	Hash Join (S,12)	[2657, 406, 93339, 924]	5886	670934	[20KB, 20KB]	1MB	45	102532655.39	
8	Seq Scan on s_riskrate_setting a	[38, 885, 2940, 983]	7725027	78594218	[612KB, 903KB]	1MB	56	275244.71	
9	Hash	[2241, 438, 2713, 381]	8536241	8536241	[1011KB, 97803KB]	10MB	18, 48	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, HashJoin becomes the performance bottleneck of the whole plan. Based on the execution time of HashJoin **[2657.406, 93339.924]** (for details about the value, see [SQL Execution Plan](#)), it can be seen that severe skew occurs on different DN's during the HashJoin operation.

In the memory information (as shown in the following figure), it can be seen that the data skew occurs in the memory usage of each node.

```
----- Memory Information (identified by plan id) -----
-----
Coordinator:
Query Peak Memory: 4MB
Datanode:
Max Query Peak Memory: 118MB
Min Query Peak Memory: 24MB
12 --Hash
Max Buckets: 131072 Max Batches: 1 Max Memory Usage: 91857kB
Min Buckets: 131072 Min Batches: 1 Min Memory Usage: 0kB
(8 rows)
```

Optimization Analysis

The preceding two symptoms indicate that this SQL statement has serious computing skew. The further lower-layer analysis on the HashJoin 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.

16.4.7 Case: Adjusting the Partial Clustering Key

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-rows
1	→ Row Adapter	1.43977.750	1	1	96 (159KB)	1			112 1097180.70
2	→ Vector Streaming (type: GATHER)	1.43977.750	1	1	96 (156KB)	1			112 1097180.70
3	→ Vector Sort	1.14348.464,14348.464	1	1	96 (169KB, 349KB)	10MB	[0, 96]		112 1097177.61
4	→ Vector Hash Aggregate	1.14348.298,14348.298	1	1	96 (169KB, 306KB)	10MB	[0, 128]		132 1097177.58
5	→ Vector Append	1.14347.522,14347.522	1	1	96 (138B, 138B)	1MB			132 1097177.42
6	→ Vector Subquery Scan on "SELECT 1"	1.10112.872,10112.872	0	32	1 (98KB, 98KB)	1MB			132 649136.85
7	→ Vector Sort Aggregate	1.10112.871,10112.871	0	32	1 (152KB, 152KB)	1MB			136 649136.84
8	→ Vector Nest Loop Semi Join (5, 13)	1.10112.889,10112.889	0	2	1 (480KB, 480KB)	1MB			136 649136.78
9	→ Vector Append	1.10112.880,10112.880	0	2	1 (29B, 29B)	1MB			136 649085.57
10	→ Partitioned Dfs Scan on pd_data_ap_app_acct_sec_trade_day	1.10112.572,10112.572	0	1	1 (2964KB, 2964KB)	1MB			87 649051.65
11	→ Vector Adapter	1.10.005,0.005	0	1	1 (114KB, 114KB)	1MB			212 89.92
12	→ Seg Scan on cstore_pg_delta_3278763770 app_acct_sec_trade_day	1.10.002,0.002	0	0	1 (197KB, 197KB)	1MB			212 89.92
13	→ Vector Materialize	1.10.01	0	2	1 (10, 01)	10MB			11 81.13
14	→ Vector Append	1.10.01	0	2	1 (10, 01)	1MB			11 81.12
15	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	1.10.01	0	1	1 (10, 01)	1MB			11 48.91
16	→ Vector Adapter	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
17	→ Seg Scan on cstore_pg_delta_4036128044 act_acct_opt_his	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
18	→ Vector Subquery Scan on "SELECT 2"	1.19.038,19.038	0	32	1 (98KB, 98KB)	1MB			132 15809.79
19	→ Vector Sort Aggregate	1.19.036,19.036	0	32	1 (152KB, 152KB)	1MB			136 15809.78
20	→ Vector Nest Loop Semi Join (21, 25)	1.18.897,18.897	0	2	1 (480KB, 480KB)	1MB			136 15809.67
21	→ Vector Append	1.18.894,18.894	0	2	1 (29B, 29B)	1MB			136 15752.51
22	→ Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_03	1.18.886,18.886	0	1	1 (149KB, 149KB)	1MB			87 15718.59
23	→ Vector Adapter	1.18.004,0.004	0	1	1 (114KB, 114KB)	1MB			212 39.92
24	→ Seg Scan on cstore_pg_delta_438785702 app_acct_sec_trade_day_03	1.18.002,0.002	0	1	1 (379KB, 379KB)	1MB			212 39.92
25	→ Vector Materialize	1.10.01	0	2	1 (10, 01)	10MB			11 81.13
26	→ Vector Append	1.10.01	0	2	1 (10, 01)	1MB			11 81.12
27	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	1.10.01	0	1	1 (10, 01)	1MB			11 48.91
28	→ Vector Adapter	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
29	→ Seg Scan on cstore_pg_delta_4036128044 act_acct_opt_his	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
30	→ Vector Subquery Scan on "SELECT 3"	1.4216.008,4216.008	1	32	1 (98KB, 98KB)	1MB			132 432236.78
31	→ Vector Sort Aggregate	1.4216.004,4216.004	1	32	1 (1791KB, 1791KB)	1MB			136 432236.78
32	→ Vector Nest Loop Semi Join (33, 37)	1.4215.608,4215.608	1	2	1 (480KB, 480KB)	1MB			136 432236.67
33	→ Vector Append	1.4212.846,4212.846	1	2	1 (29B, 29B)	1MB			136 432156.50
34	→ Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_02	1.4212.846,4212.846	1	1	1 (250KB, 250KB)	1MB			86 432151.58
35	→ Vector Adapter	1.10.005,0.005	0	1	1 (114KB, 114KB)	1MB			212 39.92
36	→ Seg Scan on cstore_pg_delta_1206972051 app_acct_sec_trade_day_02	1.10.002,0.002	0	1	1 (379KB, 379KB)	1MB			212 39.92
37	→ Vector Materialize	1.12.789,2.789	1	2	1 (250KB, 250KB)	1MB	[0, 17]		11 81.13
38	→ Vector Append	1.12.616,2.616	1	2	1 (968KB, 968KB)	1MB			11 81.12
39	→ Partitioned Dfs Scan on pd_data_act_acct_opt_his	1.12.614,2.614	1	1	1 (169KB, 169KB)	1MB			11 48.91
40	→ Vector Adapter	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
41	→ Seg Scan on cstore_pg_delta_4036128044 act_acct_opt_his	1.10.01	0	1	1 (10, 01)	1MB			11 2.21
(41 rows)									

Optimization Analysis

After further analysis, we found that the filter condition `acct_id = 'A012709548'`:
bpchar exists in the two tables.

```
--Partitioned Dfs Scan on pd_data_ap_app_acct_sec_trade_day
Filter: ((pd_data_ap_app_acct_sec_trade_day.acct_code == '588')::text AND (((CASE WHEN (pd_data_ap_app_acct_sec_trade_day.acct_code == ANY ('{'2019,2024,204'})::text))) THEN ((pd_data_ap_app_acct_sec_trade_day.buy_vol * pd
--Partitioned Dfs Scan on pd_data_act_acct_opt_his
Filter: ((pd_data_act_acct_opt_his.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Seg Scan on cstore_pg_delta_3278763770 app_acct_sec_trade_day
Filter: ((cstore_pg_delta_3278763770.app_acct_sec_trade_day.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_03
Filter: ((hd_data_ap_app_acct_sec_trade_day_03.acct_code == '588')::text AND (((CASE WHEN (hd_data_ap_app_acct_sec_trade_day_03.acct_code == ANY ('{'2019,2024,204'})::text))) THEN ((hd_data_ap_app_acct_sec_trade_day_03.buy
--Partitioned Dfs Scan on hd_data_act_acct_opt_his
Filter: ((hd_data_act_acct_opt_his.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Seg Scan on cstore_pg_delta_438785702 app_acct_sec_trade_day_03
Filter: ((cstore_pg_delta_438785702.app_acct_sec_trade_day_03.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Partitioned Dfs Scan on hd_data_ap_app_acct_sec_trade_day_02
Filter: ((hd_data_ap_app_acct_sec_trade_day_02.acct_code == '588')::text AND (((CASE WHEN (hd_data_ap_app_acct_sec_trade_day_02.acct_code == ANY ('{'2019,2024,204'})::text))) THEN ((hd_data_ap_app_acct_sec_trade_day_02.buy
--Seg Scan on cstore_pg_delta_1206972051 app_acct_sec_trade_day_02
Filter: ((cstore_pg_delta_1206972051.app_acct_sec_trade_day_02.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Partitioned Dfs Scan on pd_data_act_acct_opt_his
Filter: ((pd_data_act_acct_opt_his.acct_id = 'A012709548')::bpchar) AND (((CASE WHEN (cstore_pg_delta_4036128044.act_acct_opt_his
--Seg Scan on cstore_pg_delta_4036128044 act_acct_opt_his
Filter: ((cstore_pg_delta_4036128044.act_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.

16.4.8 Case: Adjusting the Table Storage Mode in a Medium Table

Symptom

In the GaussDB(DWS) database, row-store tables use the row execution engine, and column-store tables use the column execution engine. If both row-store table and column-store tables exist in a SQL statement, the system will automatically select the row execution engine. The performance of a column execution engine (except for the indexscan related operators) is much better than that of a row execution engine. Therefore, a column-store table is recommended. This is important for some medium result set dumping tables, and you need to select a proper table storage type.

During the test at a site, if the following execution plan is performed, the customer expects that the performance can be improved and the result can be returned within 3s.

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Streaming (type: GATHER)	4.681, 0.959	7	17	304KB	1MB			41 101740.13
2	Hash Join (8,7)	16.567, 3.849	7	17	1393KB	1MB			42 101759.10
3	Append	16.567, 3.849	7	17	1393KB	1MB			49 89969.76
4	Row Adapter	16.567, 3.849	7	17	1393KB	1MB			49 70615.19
5	Partitioned Dfs Scan on sd_data.acct_account_his ta	12288.421, 2558.707	33011417	99098596	11002KB, 1012KB	1MB			49 70615.19
6	Seq Scan on cs50xx_pg_delta_2425217623 ta	1463.977, 168.707	1740161	9010840	115KB, 159KB	1MB			80 18354.56
7	Hash	14.385, 7.997	9	32	1260KB, 292KB	16MB			30 100.17
8	Streaming (type: REDISTRIBUTE)	14.384, 7.997	9	32	11054KB, 1056KB	1MB	[0, 36]		30 100.17
9	Hash Join (10,11)	10.162, 1.043	9	32	168B, 88B	1MB			30 100.06
10	Seq Scan on pg_temp_cm_5001_140148717123328_input_acct_id_tbl tbl	10.001, 0.1765, 240.284	1740161	316260	111KB, 11KB	1MB			12 35.89
11	Hash	10.001, 0.849	9	32	1258KB, 280KB	16MB			19 80.31
12	HashAggregate	10.001, 0.849	9	32	110KB, 19KB	1MB	[0, 37]		19 80.30
13	Seq Scan on public.row_unlogged_table	10.000, 0.847	449	449	119KB, 19KB	1MB			19 78.70
(13 rows)									

Optimization Analysis

It is found that the row engine is used after analysis, because both the temporary plan table input_acct_id_tbl and the medium result dumping table row_unlogged_table use a row-store table.

After the two tables are changed into column-store tables, the system performance is improved and the result is returned by 1.6s.

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Row Adapter	1.567, 3.867	7	17	38KB				41 101758.52
2	Vector Streaming (type: GATHER)	1.567, 3.849	7	17	1393KB				42 101759.52
3	Vector Hash Join (4,8)	18.130, 1429.101	7	17	1234KB, 244KB	16MB			49 101757.48
4	Vector Append	1542.823, 1452.479	8691770	108109436	11KB, 1KB	1MB			49 89969.76
5	Partitioned Dfs Scan on sd_data.acct_account_his ta	1286.796, 1196.800	3340754	99098596	1861KB, 1012KB	1MB			49 70615.19
6	Vector Streaming (type: REDISTRIBUTE)	1286.796, 1196.800	3340754	99098596	1861KB, 1012KB	1MB			49 70615.19
7	Seq Scan on cs50xx_pg_delta_2425217623 ta	1152.598, 168.048	1741016	9010840	115KB, 159KB	1MB			80 18354.56
8	Vector Streaming (type: REDISTRIBUTE)	17.727, 12.981	9	32	11052KB, 1141KB	1MB	[0, 40]		30 118.56
9	Vector Hash Join (10,11)	10.132, 4.955	9	32	1217KB, 2217KB	16MB			30 118.45
10	CStore Scan on pg_temp_cm_5001_140148155066112_input_acct_id_tbl tbl	14.372, 4.372	1	31848	1207KB, 207KB	1MB			12 81.00
11	Vector Hash Aggregate	10.062, 0.209	9	32	1222KB, 222KB	16MB	[0, 35]		19 33.67
12	CStore Scan on public.col_unlogged_table	10.011, 0.107	449	449	1841KB, 598KB	1MB			19 32.08
(12 rows)									

16.4.9 Case: Adjusting the Local Clustering Column

Symptom

During the test at a site, if the following execution plan is performed, the customer expects that the performance can be improved and the result can be returned within 3s.

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	Row Adapter	11626.124	24	6472	840B				287 59224.48
2	Vector Streaming (type: GATHER)	11626.091	24	6472	1393KB				287 59224.49
3	Vector Hash Aggregate	111846.971, 11888.002	24	6472	13703KB, 3703KB	16MB	[302, 312]		287 591799.12
4	Vector Streaming (type: REDISTRIBUTE)	111846.955, 11887.440	63	6476	1713KB, 1100KB	1MB			287 592750.10
5	Vector Hash Join (6,10)	111046.695, 11554.757	63	6476	1351KB, 3591KB	16MB			287 591677.08
6	Vector Append	15629.989, 10799.042	108412935	114745164	11KB, 1KB	1MB			27 448209.06
7	Partitioned Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	15513.750, 3548.903	9515860	97495460	12047KB, 25647KB	1MB			27 302346.53
8	Vector Adapter	11006.758, 1165.251	13243595	13243621	1122KB, 1230B	1MB			27 47162.82
9	Seq Scan on cs50xx_pg_delta_2166320801 ss	7844.235, 631.631	13243595	13243621	1122KB, 2193B	1MB			27 47162.82
10	Vector Streaming (type: REDISTRIBUTE)	1941.659, 979.940	268	2244	11310KB, 11310KB	1MB			338 70454.93
11	Vector Hash Join (11,48)	1784.245, 964.821	24	187	13947KB, 3047KB	16MB			338 70455.28
12	Vector Append	1784.245, 964.820	24	187	13910KB, 3050KB	16MB			312 69338.06
13	Vector Append	1587.304, 521.920	3995798	3995816	11KB, 29B	1MB			87 22527.08
14	Partitioned Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	1387.594, 531.472	3995798	3995798	1218KB, 2190KB	1MB			87 22516.98
15	Vector Adapter	10.003, 0.008	0	120	1130KB, 1130B	1MB			148 10.10
16	Seq Scan on cs50xx_pg_delta_360882137 ss	10.001, 0.004	0	120	1130KB, 1130B	1MB			148 10.10
17	Vector Streaming (type: REDISTRIBUTE)	1398.303, 414.767	288	2244	1379KB, 3790B	1MB	[352, 388]		224 43811.56
18	Vector Hash Join (15,23)	189.559, 410.302	24	187	1272KB, 2517KB	16MB			224 43755.80
19	Vector Append	118.727, 130.776	2342168	4016127	11KB, 1KB	1MB			42 3757.78
20	Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	118.720, 130.588	2342168	4016127	1551KB, 1100KB	1MB			42 3757.68
21	Vector Adapter	10.002, 0.022	0	120	1130KB, 2930B	1MB			90 10.10
22	Seq Scan on cs50xx_pg_delta_1125056894 ss	10.001, 0.020	0	120	1130KB, 1200B	1MB			90 10.10
23	Vector Hash Join (24, 28)	189.695, 240.830	24	61	12493KB, 2780KB	16MB	[190, 202]		139 37472.66
24	Vector Append	1140.386, 144.540	2064690	2478869	11KB, 29B	1MB			79 4375.60
25	Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	1140.386, 144.540	2064690	2478869	11450KB, 11450KB	1MB			79 4375.60
26	Vector Adapter	10.003, 0.008	0	120	1130KB, 6790B	1MB			128 10.10
27	Seq Scan on cs50xx_pg_delta_360924216 ss	10.001, 0.021	0	120	1130KB, 1490B	1MB			128 10.10
28	Vector Streaming (type: REDISTRIBUTE)	119.877, 94.215	24	61	1613KB, 1610KB	1MB	[160, 172]		148 31544.18
29	Vector Hash Join (30,34)	119.793, 76.829	24	61	12570KB, 2460KB	16MB			148 31544.82
30	Vector Append	139.193, 39.191	24	3760770	11KB, 29B	1MB			44 6953.93
31	Seq Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	139.194, 39.184	24	3760770	11450KB, 11450KB	1MB			44 6953.93
32	Vector Adapter	10.003, 0.008	0	120	1130KB, 6790B	1MB			90 10.10
33	Seq Scan on cs50xx_pg_delta_2166320801 ss	10.001, 0.011	0	120	1130KB, 1490B	1MB			90 10.10
34	Vector Hash Join (35,39)	118.617, 48.471	1	4	12413KB, 2503KB	16MB	[0, 102]		148 23574.24
35	Vector Append	19.048, 9.048	1	1249301	11KB, 1KB	1MB			89 4485.80
36	Partitioned Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	19.036, 9.036	1	1249301	17490KB, 27490KB	1MB			89 4485.80
37	Vector Adapter	10.011, 0.011	0	1	1130KB, 790B	1MB			89 1.06
38	Seq Scan on cs50xx_pg_delta_1491921217 ss	10.009, 0.009	0	1	1130KB, 1130B	1MB			89 1.06
39	Vector Hash Join (43,44)	119.329, 34.149	1	4	12140KB, 2362KB	16MB	[0, 83]		87 18995.78
40	Vector Append	119.344, 10.864	1	1182922	1130B, 1130B	1MB			29 4485.24
41	Partitioned Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	119.344, 10.864	1	1182922	1130KB, 8590KB	1MB			29 4485.14
42	Vector Adapter	10.011, 0.011	0	1	1130KB, 3090B	1MB			30 1.10
43	Seq Scan on cs50xx_pg_delta_345394125 ss	10.005, 0.001	0	1	1130KB, 1130B	1MB			30 1.10
44	Vector Append	119.480, 24.149	1	2	1130KB, 1130B	1MB	[0, 40]		28 11492.16
45	Partitioned Dfs Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	119.184, 28.893	1	1	1130KB, 27490KB	1MB			28 11492.42
46	Vector Adapter	10.257, 0.389	0	1	1130KB, 3090B	1MB			28 62.74
47	Seq Scan on cs50xx_pg_delta_489619337 ss	10.215, 0.389	0	1	1130KB, 1490B	1MB			28 62.74
48	CStore Scan on lfbank2_f_ev_dp_kdpl_zhminx ss	10.072, 0.190	708	708	1110KB, 1110KB	1MB	[0, 40]		34 1521.06
(48 rows)									

Optimization Analysis

The analysis shows that the performance bottleneck of this plan is lfbank2_f_ev_dp_kdpl_zhminx. The scan condition of this table is as follows:

Predicate Information (identified by plan id)									
5	--Vector Hash Join (6,10)								
	Hash Cond: (((mx.zhanghao)::text = (dd.zhanghao)::text) AND ((mx.farendma)::text = (cf.farendma)::text))								
7	--Partitioned Dfs Scan on lfbank_f_ev_dp_kdpl_zhminx mx								
	Pushdown Predicate Filter: (((mx.yezdminc)::text = 'DPLDGBAL'::text))								
9	--Seq Scan on cstore.pg_delta_z186z00u1 mx								
	Filter: ((mx.yezdminc)::text = 'DPLDGBAL'::text)								
	Rows Removed by Filter: 489								

Try to modify the lfbank.f_ev_dp_kdpl_zhmin table to a column-store table, and then create the PCK (local clustering) in the yezdminc column, and set **PARTIAL_CLUSTER_ROWS** to 10000000. The execution plan after optimization is as follows:

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-- Row Adapter	2493.864	24	5494	3549B				287 745538.94
2	--> Vector Streaming (type: GATHER)	2493.847	24	5494	1913KB				287 745538.94
3	--> Vector Hash Aggregate	(1597.746,2099.012)	24	5494	(1705KB, 3703KB)	1MB	[300,318]		287 744921.76
4	--> Vector Streaming(type: REDISTRIBUTE)	(1597.246,2099.032)	63	5494	(1778KB, 1209KB)	1MB			287 744872.82
5	--> Vector Hash Join (6,7)	(2317.650,2479.801)	63	5494	(1803KB, 3003KB)	10MB			287 744799.18
6	--> CStore Scan on lfbank.f_ev_dp_kdpl_zhminx_column mx	(2317.650,2479.801)	63	5494	(1803KB, 1209KB)	1MB			28 500296.59
7	--> Vector Streaming(type: BROADCAST)	(1715.439,725.474)	24	2244	(1358KB, 1358KB)	1MB	[419,423]		287 81239.75
8	--> Vector Hash Join (9,42)	(4610.787,723.121)	24	287	(1047KB, 3047KB)	10MB			287 81189.11
9	--> Vector Hash Join (10,11)	(4610.187,722.433)	24	287	(1287KB, 2907KB)	10MB			216 79874.88
10	--> CStore Scan on lfbank.f_ev_dp_kdpl_zhminx mx	(1246.145,248.863)	3995798	(1800KB, 3915KB)	1MB				89 33645.98
11	--> Vector Streaming(type: BROADCAST)	(443.942,448.609)	288	2244	(979KB, 979KB)	1MB	[252,268]		226 43810.56
12	--> Vector Hash Join (13,17)	(590.444,440.531)	24	287	(1732KB, 2917KB)	10MB			226 43750.50
13	--> Vector Append	(20.159,146.517)	2342145	4014237	(119B, 19B)	1MB			42 3797.78
14	--> Dfs Scan on lfbank2.f_ev_dp_kdpl_zhminx dd	(20.151,146.194)	2342145	4014217	(163KB, 1104KB)	1MB			42 3787.68
15	--> Vector Adapter	(0.001,0.001)	0	120	(139KB, 139KB)	1MB			99 10.10
16	--> Seq Scan on cstore.pg_delta_1120506844 dd	(0.001,0.002)	0	120	(139KB, 139KB)	1MB			99 10.10
17	--> Vector Hash Right Join (16,22)	(190.411,291.664)	24	51	(1659KB, 2789KB)	10MB	[190,205]		159 37472.46
18	--> Vector Append	(148.194,187.043)	2064690	2478959	(129B, 29B)	1MB			73 4375.60
19	--> Dfs Scan on lfbank2.f_ev_vou_cnd_kdpl_zhminx dd	(148.049,186.905)	2064690	2477939	(1145KB, 1168KB)	1MB			73 4363.49
20	--> Vector Adapter	(0.001,0.018)	0	120	(179KB, 97KB)	1MB			128 10.10
21	--> Seq Scan on cstore.pg_delta_309246256 dd	(0.001,0.003)	0	120	(149KB, 149KB)	1MB			128 10.10
22	--> Vector Streaming(type: REDISTRIBUTE)	(190.054,94.368)	24	51	(1619KB, 866KB)	1MB	[160,172]		148 31645.18
23	--> Vector Hash Join (24,25)	(129.849,40.488)	24	51	(1817KB, 248KB)	10MB			148 31644.82
24	--> Vector Append	(38.109,38.109)	24	3704070	(129B, 29B)	1MB			46 6084.93
25	--> Dfs Scan on lfbank2.f_ev_dp_kdpl_zhminx dd	(38.102,38.102)	24	3703950	(1481KB, 1481KB)	1MB			46 6044.89
26	--> Vector Adapter	(0.001,0.003)	0	120	(179KB, 97KB)	1MB			99 10.10
27	--> Seq Scan on cstore.pg_delta_226997238 dd	(0.002,0.002)	0	120	(149KB, 149KB)	1MB			99 10.10
28	--> Vector Hash Join (29,33)	(119.374,46.091)	1	4	(1414KB, 2603KB)	10MB	[0,182]		144 23174.24
29	--> Vector Append	(8.778,8.778)	1	1294301	(119B, 139B)	1MB			89 5459.50
30	--> Partitioned Dfs Scan on lfbank2.f_ev_per_kdpl_zhminx dd	(8.768,8.768)	1	1294301	(794KB, 794KB)	1MB			89 5459.44
31	--> Vector Adapter	(0.001,0.003)	0	1	(740KB, 740KB)	1MB			89 1.06
32	--> Seq Scan on cstore.pg_delta_1491992217 dd	(0.001,0.008)	0	1	(139KB, 139KB)	1MB			89 1.06
33	--> Vector Hash Join (34,38)	(19.310,34.094)	1	8	(2249KB, 2382KB)	10MB	[0,85]		87 14908.78
34	--> Vector Append	(151.050,11.000)	1	1182322	(139B, 19B)	1MB			29 4484.24
35	--> Partitioned Dfs Scan on lfbank2.f_ev_per_kdpl_zhminx dd	(10.988,10.988)	1	1182321	(809KB, 809KB)	1MB			29 4483.14
36	--> Vector Adapter	(0.001,0.003)	0	1	(139KB, 139KB)	1MB			30 1.10
37	--> Seq Scan on cstore.pg_delta_849394125 dd	(0.001,0.008)	0	1	(139KB, 139KB)	1MB			30 1.10
38	--> Vector Append	(19.285,28.768)	1	2	(119B, 19B)	1MB	[0,40]		28 11682.16
39	--> Partitioned Dfs Scan on lfbank2.f_ev_per_kdpl_zhminx dd	(19.059,28.244)	1	1	(1693KB, 1749KB)	1MB			28 11659.42
40	--> Vector Adapter	(0.250,0.539)	0	1	(139KB, 139KB)	1MB			28 62.74
41	--> Seq Scan on cstore.pg_delta_480619537 dd	(0.247,0.536)	0	1	(149KB, 149KB)	1MB			28 62.74
42	--> CStore Scan on lfbank2.f_ev_dp_kdpl_zhminx mx	(0.136,0.538)	708	708	(1139KB, 1139KB)	1MB	[48,48]		36 1331.06
(42 rows)									

NOTE

- This method actually sacrifices the performance during data import to improve the query performance.
- The number of local sorting tuples is increased, and you need to increase the value of **psort_work_mem** to improve the sorting efficiency.

16.4.10 Case: Reconstructing Partition Tables

Symptom

In the following simple SQL statements, the performance bottlenecks exist in the scan operation of dwcjk.

postgres=# explain performance select zqdh, count(1) from dwcjk where cjrq = '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 771106.93
2	--> Vector Streaming (type: GATHER)	1599.781	58	12	210KB				7 771106.93
3	--> Vector Hash Aggregate	(1444.092,1446.332)	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.dwcjk	(330.178,1021.695)	10000000	1623137	(786KB, 786KB)	1MB			7 120402.00
(6 rows)									

Optimization Analysis

Obviously, there are date features in the cjrq field of table data in the service layer, and this meet the features of a partitioned table. Replan the table definition of the dwcjk table. Set the cjrq field as a partition key, and day as an interval unit. Define the partitioned table dwcjk_part. The modified result is as follows, and the performance is nearly doubled.

```
postgres=# explain performance select zqdh, count(1) from dwobj_part where cjqrq = '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 | [162.145, 515.752] | 340 | 14 | [2297KB, 2297KB] | 16MB | [16, 16] | 7 | 128857.14
6 | -> Vector Partition Iterator | [162.630, 275.590] | 10000000 | 1691000 | [312KB, 312KB] | 1MB | | 7 | 120402.00
7 | -> Partitioned CStore Scan on public.dwobj_part | [161.746, 275.207] | 10000000 | 1691000 | [795KB, 795KB] | 1MB | | 7 | 120402.00
(7 rows)
```

16.4.11 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 column 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 results aggregated from the lower-layer is correct aggregation results, and the upper-level operator can be directly used, as shown in the following figure:

```
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(DWS) 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**, and **3**.

- When the value is set to **1**, the first plan is forcibly generated.
- When the value 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 value 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 value 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.

```
set best_agg_plan to 1;
SET
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)
set best_agg_plan to 2;
SET
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)
set best_agg_plan to 3;
SET
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.

16.4.12 Case: Rewriting SQL and Deleting Subqueries (Case 1)

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:

```

postgres=# explain select 1,(select count(*)
postgres#         from customer_address_001 a4
postgres#         where a4.ca_address_sk = a.ca_address_sk
postgres#         ) 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*.

16.4.13 Case: Rewriting SQL and Deleting Subqueries (Case 2)

Symptom

On a site, the customer gave the feedback saying that the execution time of the following SQL statements lasted over one day and did not end:

```
UPDATE calc_empfyc_c_cusr1 t1
SET ln_rec_count =
(
SELECT CASE WHEN current_date - ln_process_date + 1 <= 12 THEN 0 ELSE t2.ln_rec_count END
FROM calc_empfyc_c1_policysend_tmp t2
WHERE t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 = t2.ls_policyno_cusr1
)
WHERE dsign = '1'
AND flag = '1'
AND EXISTS
(SELECT 1
FROM calc_empfyc_c1_policysend_tmp t2
WHERE t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 = t2.ls_policyno_cusr1
);
```

The corresponding execution plan is as follows:

```
Streaming (type: GATHER) (cost=44693.26..19548919558.34 rows=4058158 width=1061)
Node/s: All datanodes
--> Update on channel.calc_empfyc_c_cusr1 t1 (cost=44693.26..19548919558.34 rows=4058158 width=1061)
--> Hash Join (cost=44693.26..19548919558.34 rows=4058158 width=1061)
--> Hash Cond: (((t1.ln_branch)::text = (t2.ln_branch)::text)) AND (((t1.ls_policyno_cusr1)::text = (t2.ls_policyno_cusr1)::text))
--> Seq Scan on channel.calc_empfyc_c_cusr1 t1 (cost=0.00..28692.39 rows=7105667 width=1055)
--> Filter: ((t1.dsign = '1')::bool) AND (t1.flag = '1')::bool
--> Hash (cost=2112.06..2112.16 rows=108998016 width=37)
--> Unique (cost=2112.06..2112.16 rows=108998016 width=37)
--> Sort (cost=2112.06..2112.09 rows=775 width=37)
--> Sort Key: ((t2.ln_branch)::text), ((t2.ls_policyno_cusr1)::text)
--> Streaming (type: BROADCAST) (cost=2109.81..2111.85 rows=775 width=37)
--> Spawn on: All datanodes
--> HashAggregate (cost=2109.81..2109.82 rows=12 width=37)
--> Group By Key: (t2.ln_branch)::text, (t2.ls_policyno_cusr1)::text
--> Seq Scan on channel.calc_empfyc_c1_policysend_tmp t2 (cost=0.00..1406.87 rows=1703094 width=37)
SubPlan 1
--> Result (cost=0.00..308262.89 rows=108998016 width=44)
--> Filter: (((t1.ln_branch)::text = (t2.ln_branch)::text) AND ((t1.ls_policyno_cusr1)::text = (t2.ls_policyno_cusr1)::text))
--> Materialize (cost=0.00..295489.68 rows=108998016 width=44)
--> Streaming (type: BROADCAST) (cost=0.00..286974.21 rows=108998016 width=44)
--> Spawn on: All datanodes
--> Seq Scan on channel.calc_empfyc_c1_policysend_tmp t2 (cost=0.00..1406.87 rows=1703094 width=44)
```

Optimization

SubPlan exists in the execution plan, and the calculation accounts for a large proportion in the SubPlan query. That is, SubPlan is a performance bottleneck.

Based on the SQL syntax, you can rewrite the SQL statements and delete SubPlan as follows:

```
UPDATE calc_empfyc_c_cusr1 t1
SET ln_rec_count = CASE WHEN current_date - ln_process_date + 1 <= 12 THEN 0 ELSE t2.ln_rec_count END
FROM calc_empfyc_c1_policysend_tmp t2
WHERE
t1.dsign = '1' AND t1.flag = '1'
AND t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 = t2.ls_policyno_cusr1;
```

The modified SQL statement task is complete within 50s.

16.4.14 Case: Rewriting SQL Statements and Eliminating Prune Interference

Symptom

In a test at a site, **ddw_f10_op_cust_asset_mon** is a partitioned table and the partition 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 SQL Scan table 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 the `pg_proc`. Based on the function behaviors described in PostgreSQL, this type of function cannot be

converted into the Const value in the preprocessing phase, which is the root cause of preventing partition pruning.

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.

16.4.15 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');
```

Some special usages are as follows:

```
SELECT
  ls_pid_cusr1,COALESCE(max(round((current_date-bthdate)/365)),0)
FROM calc_empfyc_c1_result_tmp t1,p10_md_tmp_t2 t2
WHERE t1.ls_pid_cusr1 = any(values(id),(id15))
GROUP BY ls_pid_cusr1;
```

Where **id** and **id15** are columns of p10_md_tmp_t2. **ls_pid_cusr1 = any(values(id), (id15))** equals **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(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..820714640.69 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.mmdes
            -> Materialize (cost=0.00..147.29 rows=252608 width=17)
              Output: t1.ls_pid_cusr1
              -> Streaming(type: BROADCAST) (cost=0.00..127.56 rows=252608 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 W=VALUES* (cost=0.00..0.01 rows=64 width=38)
              Output: W=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 hashjoin. 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
        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
        WHERE t1.ls_pid_cusr1 = id15
    )
)
GROUP BY ls_pid_cusr1;
```

The optimized SQL queries consist of two equivalent join subqueries, and each subquery can be used for hashjoin in this scenario. The optimized execution plan is as follows:

id	operation	A-time	A-rows	B-rows	Peak Memory	Memory	A-width
1	-> Streaming (type: GATHER)	6737.281	0	192	292KB		
2	-> Insert on channel.calc_empfyc_c1_result_age_tmp	[4665.024,4990.666]	0	192	[1108KB, 1108KB]	1MB	
3	-> HashAggregate	[4664.996,4990.641]	0	192	[12KB, 12KB]	10MB	
4	-> Streaming (type: BROADCAST)	[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 (8,9)	[2011.226,3080.697]	0	3947	[6KB, 6KB]	1MB	
8	-> Seq Scan on channel.p10_md_tmp_t2 t2	[803.782,1238.984]	443525717	443523360	[12KB, 12KB]	1MB	
9	-> Hash	[4.357,328.979]	252608	252608	[482KB, 482KB]	16MB	[38, 39]
10	-> Streaming (type: BROADCAST)	[2.345,326.320]	252608	252608	[2090KB, 2090KB]	1MB	
11	-> Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1	[0.011,0.030]	3947	3947	[11KB, 11KB]	1MB	
12	-> Hash Join (13,14)	[1376.258,2066.110]	0	64	[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	[38, 37]
15	-> Streaming (type: BROADCAST)	[1.276,1.868]	252608	252608	[2090KB, 2090KB]	1MB	
16	-> Seq Scan on channel.calc_empfyc_c1_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.

16.4.16 Case: Setting Partial Cluster Keys

You can add **PARTIAL CLUSTER KEY**(*column_name*[,...]) to the definition of a column-store table to set one or more columns of this table as partial cluster keys. In this way, each 70 CUs (4.2 million rows) will be sorted based on the cluster keys by default during data import and the value range is narrowed down for each of the new 70 CUs. If the **where** condition in the query statement contains these columns, the filtering performance will be improved.

1. Use partial cluster keys.

```
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
)
with (orientation = column)
distribute by hash(L_ORDERKEY);

select
sum(l_extendedprice * l_discount) as revenue
from
lineitem
where
l_shipdate >= '1994-01-01'::date
and l_shipdate < '1994-01-01'::date + interval '1 year'
and l_discount between 0.06 - 0.01 and 0.06 + 0.01
and l_quantity < 24;

```

In the **where** condition, both the **l_shipdate** and **l_quantity** columns have a few distinct values, and their values can be used for min/max filtering. Therefore, modify the table definition 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
, partial cluster key(l_shipdate, l_quantity)
)
with (orientation = column)
distribute by hash(L_ORDERKEY);

```

Import the data again and run the query statement. Then, compare the execution time before and after partial cluster keys are used.

Figure 16-13 Partial cluster keys not used

id	operation	A-time	A-rows	E-rows	Peak Memory	A-width	E-width	E-costs
1	-> Row Adapter	1653.156	1	1	12KB		44	205803.90
2	-> Vector Aggregate	1653.146	1	1	184KB		44	205803.90
3	-> Vector Streaming (type: GATHER)	1653.070	1	1	174KB		44	205803.90
4	-> Vector Aggregate	[1481.497,1481.497]	1	1	[225KB, 225KB]		44	205803.84
5	-> CStore Scan on public.lineitem	[1405.004,1405.004]	114160	111485	[792KB, 792KB]		12	205246.40

(5 rows)

Figure 16-14 CU loading without partial cluster keys

```

5 --CStore Scan on public.lineitem
  datanode1 (actual time=40.623..1405.004 rows=114160 loops=1)
    datanode1 (RoughCheck CU: CUNone: 0, CUSome: 101)
    datanode1 (LLVM Optimized)
    datanode1 (Buffers: shared hit=18385 read=23)
    datanode1 (CPU: ex c/r=31917, ex cyc=3643646206, inc cyc=3643646206)

```

Figure 16-15 Partial cluster keys used

id	operation	A-time	A-rows	E-rows	Peak Memory	A-width	E-width	E-costs
1	-> Row Adapter	459.539	1	1	12KB		44	205693.85
2	-> Vector Aggregate	459.528	1	1	184KB		44	205693.85
3	-> Vector Streaming (type: GATHER)	459.452	1	1	174KB		44	205693.85
4	-> Vector Aggregate	[285.177,285.177]	1	1	[225KB, 225KB]		44	205693.79
5	-> CStore Scan on public.lineitem	[249.757,249.757]	114160	89475	[792KB, 792KB]		12	205246.40

Figure 16-16 CU loading with partial cluster keys

```

5 --CStore Scan on public.lineitem
  datanode1 (actual time=23.017..249.757 rows=114160 loops=1)
    datanode1 (RoughCheck CU: CUNone: 84, CUSome: 17)
    datanode1 (LLVM Optimized)
    datanode1 (Buffers: shared hit=2853 read=23)
    datanode1 (CPU: ex c/r=5673, ex cyc=647656146, inc cyc=647656146)

```

After partial cluster keys are used, the execution time of **5-- CStore Scan on public.lineitem** decreases by 1.2s because 84 CUs are filtered out.

2. Select partial cluster keys.
 - The following data types support cluster keys: character varying(n), varchar(n), character(n), char(n), text, nvarchar2, timestamp with time zone, timestamp without time zone, date, time without time zone, and time with time zone.
 - Smaller number of distinct values in a partial cluster key generates higher filtering performance.
 - Columns that can filter out larger amount of data is preferentially selected as partial cluster keys.
 - If multiple columns are selected as partial cluster keys, the columns are used in sequence to sort data. You are advised to select a maximum of three columns.
3. Modify parameters to reduce the impact of partial cluster keys on the import performance.

After partial cluster keys are used, data will be sorted when they are imported, affecting the import performance. If all the data can be sorted in the memory, the keys have little impact on import. If some data cannot be sorted in the memory and is written into a temporary file for sorting, the import performance will be greatly affected.

The memory used for sorting is specified by the **psort_work_mem** parameter. You can set it to a larger value so that the sorting has less impact on the import performance.

The volume of data to be sorted is specified by the **PARTIAL_CLUSTER_ROWS** parameter of the table. Decreasing the value of this parameter reduces the amount of data to be sorted at a time. **PARTIAL_CLUSTER_ROWS** is usually used along with the **MAX_BATCHROW** parameter. The value of

PARTIAL_CLUSTER_ROWS must be an integer multiple of the **MAX_BATCHROW** value. **MAX_BATCHROW** specifies the maximum number of rows in a CU.

16.5 Troubleshooting Errors Using SQL Statements

16.5.1 Low Query Efficiency

A query task that used to take a few milliseconds to complete is now requiring several seconds, and that used to take several seconds is now requiring even half an hour. This section describes how to analyze and rectify such low efficiency issues.

Procedure

Perform the following procedure to locate the cause of this fault.

Step 1 Run the **analyze** command to analyze the database.

The **analyze** command updates data statistics information, such as data sizes and attributes in all tables. This is a lightweight command and can be executed frequently. If the query efficiency is improved or restored after the command execution, the autovacuum process does not function well and requires further analysis.

Step 2 Check whether the query statement returns unnecessary information.

For example, if we only need the first 10 records in a table but the query statement searches all records in the table, the query efficiency is fine for a table containing only 50 records but very low for a table containing 50,000 records.

If an application requires only a part of data information but the query statement returns all information, add a LIMIT clause to the query statement to restrict the number of returned records. In this way, the database optimizer can optimize space and improve query efficiency.

Step 3 Check whether the query statement still has a low response even when it is solely executed.

Run the query statement when there are no or only a few other query requests in the database, and observe the query efficiency. If the efficiency is high, the previous issue is possibly caused by a heavily loaded host in the database system or an inefficient execution plan.

Step 4 Check the same query statement repeatedly to check the query efficiency.

One major cause that will reduce query efficiency is that the required information is not cached in the memory or is replaced by other query requests because of insufficient memory resources.

Run the same query statement repeatedly. If the query efficiency increases gradually, the previous issue might be caused by this reason.

----End

16.5.2 "failed to find conversion function from unknown to text" Is Displayed When a User Executes the SELECT Statement

Description

An error message is displayed when the **SELECT** statement is executed.

failed to find conversion function from unknown to text

```
ERROR: failed to find conversion function from unknown to text
```

Possible Causes

The type of a constant in a subquery of the **SELECT** statement is not specified.

Troubleshooting Method

Convert the constant type of the subquery to a fixed type.

Example:

```
create table t1(col1 int, col2 int);

SELECT distinct Q.*
  FROM (SELECT col1,
               '2015-03-01' AS start_time,
               '2015-03-01' AS last_time
        FROM t1)Q;
```

Change it as follows:

```
SELECT distinct Q.*
  FROM (SELECT col1,
               cast('2015-03-01' as text) AS start_time,
               cast('2015-03-01' as text) AS last_time
        FROM t1)Q;
```

16.5.3 DROP TABLE Fails to Be Executed

Problem

DROP TABLE fails to be executed in the following scenarios:

- A user runs the **\dt+** command using **gsql** and finds that the *table_name* table does not exist. When the user runs the **CREATE TABLE table_name** command, an error message indicating that the *table_name* table exists is displayed. When the user runs the **DROP TABLE table_name** command, an error message indicating that the *table_name* table does not exist is displayed. In this case, the *table_name* table cannot be recreated.
- A user runs the **\dt+** command using **gsql** and finds that the *table_name* table exists in the database. When the user runs the **DROP TABLE table_name** command, an error message indicating that the *table_name* table does not exist is displayed. In this case, the *table_name* table cannot be recreated.

Possible Causes

The `table_name` table exists on some nodes only.

Troubleshooting Method

In the preceding scenarios, if **DROP TABLE** `table_name` fails to be executed, run **DROP TABLE IF EXISTS** `table_name` to successfully drop `table_name`.

16.5.4 Different Data Is Displayed for the Same Table Queried By Multiple Users

Problem

Two users log in to the same database `human_resource` and run the **select count(*) from areas** statement separately to query the `areas` table, but obtain different results.

Possible Causes

Check whether the two users really query the same table. In a relational database, a table is identified by three elements: **database**, **schema**, and **table**. In this issue, **database** is `human_resource` and **table** is `areas`. Then, check **schema**. Log in as users `dbadmin` and `user01` separately. It is found that **search_path** is **public** for `dbadmin` and `$user` for `user01`. By default, a schema having the same name as user `dbadmin`, the cluster administrator, is not created. That is, all tables will be created in **public** if no schema is specified. However, when a common user, such as `user01`, is created, the same-name schema (`user01`) is created by default. That is, all tables are created in `user01` if the schema is not specified. In conclusion, both the two users are operating the table, causing that the same-name table is not really the same table.

Troubleshooting Method

Use `schema.table` to determine a table for query.

16.5.5 An Error Occurs During the Integer Conversion

Problem

The following error is reported during the integer conversion:

```
Invalid input syntax for integer: "13."
```

Possible Causes

Some data types cannot be converted to the target data type.

Troubleshooting

Gradually narrow down the range of SQL statements to locate the fault.

16.5.6 Automatic Retry upon SQL Statement Execution Errors

With automatic retry (referred to as CN retry), GaussDB(DWS) retries an SQL statement when the execution of this statement fails. If an SQL statement sent from the **gsql** client, JDBC driver, or ODBC driver fails to be executed, the CN can automatically identify the error reported during execution and re-deliver the task to retry.

The restrictions of this function are as follows:

- Functionality restrictions:
 - CN retry increases execution success rate but does not guarantee success.
 - CN retry is enabled by default. In this case, the system records logs about temporary tables. If it is disabled, the system will not record the logs. Therefore, do not repeatedly enable and disable CN retry when temporary tables are used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
 - CN retry is enabled by default. In this case, the **unlogged** keyword is ignored in the statement for creating unlogged tables and thereby ordinary tables will be created by using this statement. If CN retry is disabled, the system records logs about unlogged tables. Therefore, do not repeatedly enable and disable CN retry when unlogged tables are used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
 - When GDS is used to export data, CN retry is supported. The existing mechanism checks for duplicate files and deletes duplicate files during data export. Therefore, you are advised not to repeatedly export data for the same foreign table unless you are sure that files with the same name in the data directory need to be deleted.
- Error type restrictions:

Only the error types in [Table 16-3](#) are supported.
- Statement type restrictions:

Support single-statement CN retry, stored procedures, functions, and anonymous blocks. Statements in transaction blocks are not supported.
- Statement restrictions of a stored procedure:
 - If an error occurs during the execution of a stored procedure containing EXCEPTION (including statement block execution and statement execution in EXCEPTION), the stored procedure can be retried. If the error is captured by EXCEPTION, the stored procedure cannot be retried.
 - Packages that use global variables are not supported.
 - **DBMS_JO** is not supported.
 - **UTL_FILE** is not supported.
- Cluster status restrictions:
 - Only DNs or GTMs are faulty.
 - The cluster can be recovered before the number of CN retries reaches the allowed maximum (controlled by [max_query_retry_times](#)). Otherwise, CN retry may fail.
 - CN retry is not supported during scale-out.

- Data import restrictions:
 - The **COPY FROM STDIN** statement is not supported.
 - The **gsql \copy from** metacommand is not supported.
 - **JDBC CopyManager copyIn** is not supported.

Table 16-3 lists the error types supported by CN retry and the corresponding error codes. You can use the GUC parameter [retry_ecode_list](#) to set the list of error types supported by CN retry. You are not advised to modify this parameter. To modify it, contact the technical support.

Table 16-3 Error types supported by CN retry

Error Type	Error Code	Remarks
CONNECTION_RESET_BY_PEER	YY001	TCP communication errors: Connection reset by peer (communication between the CN and DNs)
STREAM_CONNECTION_RESET_BY_PEER	YY002	TCP communication errors: Stream connection reset by peer (communication between DNs)
LOCK_WAIT_TIMEOUT	YY003	Lock wait timeout
CONNECTION_TIMED_OUT	YY004	TCP communication errors: Connection timed out
SET_QUERY_ERROR	YY005	Failed to deliver the SET command: Set query
OUT_OF_LOGICAL_MEMORY	YY006	Failed to apply for memory: Out of logical memory
SCTP_MEMORY_ALLOC	YY007	SCTP communication errors: Memory allocate error
SCTP_NO_DATA_IN_BUFFER	YY008	SCTP communication errors: SCTP no data in buffer
SCTP_RELEASE_MEMORY_CLOSE	YY009	SCTP communication errors: Release memory close
SCTP_TCP_DISCONNECT	YY010	SCTP communication errors: TCP disconnect
SCTP_DISCONNECT	YY011	SCTP communication errors: SCTP disconnect
SCTP_REMOTE_CLOSE	YY012	SCTP communication errors: Stream closed by remote

Error Type	Error Code	Remarks
SCTP_WAIT_POLL_UNKNOW	YY013	Waiting for an unknown poll: SCTP wait poll unknown
SNAPSHOT_INVALID	YY014	Snapshot invalid
ERRCODE_CONNECTION_RECEIVE_WRONG	YY015	Connection receive wrong
OUT_OF_MEMORY	53200	Out of memory
CONNECTION_FAILURE	08006	GTM errors: Connection failure
CONNECTION_EXCEPTION	08000	Failed to communicate with DN's due to connection errors: Connection exception
ADMIN_SHUTDOWN	57P01	System shutdown by administrators: Admin shutdown
STREAM_REMOTE_CLOSE_SOCKET	XX003	Remote socket disabled: Stream remote close socket
ERRCODE_STREAM_DUPLICATE_QUERY_ID	XX009	Duplicate query id
ERRCODE_STREAM_CONCURRENT_UPDATE	YY016	Stream concurrent update
ERRCODE_LLVM_BAD_ALLOC_ERROR	CG003	Memory allocation error: Allocate error
ERRCODE_LLVM_FATAL_ERROR	CG004	Fatal error

To enable CN retry, set the following GUC parameters:

- Mandatory GUC parameters (required by both CNs and DN's)
max_query_retry_times. For details, see [max_query_retry_times](#).

 CAUTION

If CN retry is enabled, temporary table data is logged. For data consistency, do not switch the enabled/disabled status (for details, see [max_query_retry_times](#)) for CN retry when the temporary tables are being used by sessions.

17 Stored Procedures

17.1 Data Types

A data type refers to a value set and an operation set defined on the value set. A GaussDB(DWS) database consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. GaussDB(DWS) uses corresponding functions to perform operations on data based on data types. For example, GaussDB(DWS) can perform addition, subtraction, multiplication, and division operations on data of numeric values.

17.2 Data Type Conversion

Certain data types in the database support implicit data type conversions, such as assignments and parameters invoked by functions. For other data types, you can use the type conversion functions provided by GaussDB(DWS), such as the CAST function, to forcibly convert them.

Table 17-1 lists common implicit data type conversions in GaussDB(DWS).

NOTICE

The valid value range of DATE supported by GaussDB(DWS) is from 4713 B.C. to 294276 A.D.

Table 17-1 Implicit data type conversions

Raw Data Type	Target Data Type	Remarks
CHAR	VARCHAR2	-
CHAR	NUMBER	Raw data must consist of digits.

Raw Data Type	Target Data Type	Remarks
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	-

17.3 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;
```

Its 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(DWS), 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(DWS) prefers the array type defined in a stored procedure to declare array variables.

GaussDB(DWS) supports the access of contents in an array by using parentheses, and the **extend**, **count**, **first**, and **last** functions.

NOTE

If the stored procedure contains the DML statement (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.
CREATE OR REPLACE PROCEDURE array_proc
AS
    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;
    DBMS_OUTPUT.PUT_LINE(ARRINT.COUNT);
    DBMS_OUTPUT.PUT_LINE(ARRINT(1));
    DBMS_OUTPUT.PUT_LINE(ARRINT(10));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.FIRST));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.last));
END;
/

-- Invoke the stored procedure.
CALL array_proc();

-- Delete the stored procedure.
DROP PROCEDURE array_proc;
```

17.4 record

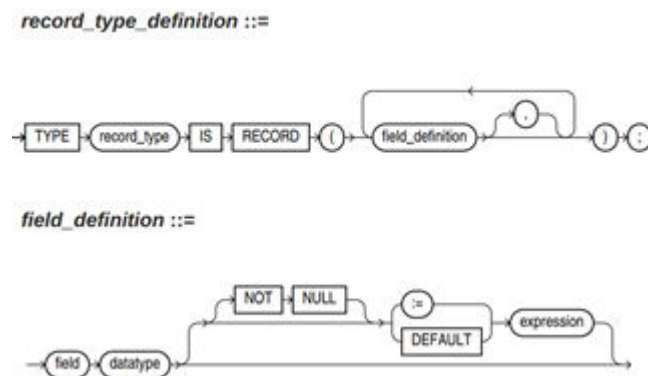
record Variables

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 17-1](#).

Figure 17-1 Syntax of the record type

The syntax is described as follows:

- **record_type**: record name
- **field**: record columns
- **datatype**: record data type
- **expression**: expression for setting a default value

NOTE

In GaussDB(DWS):

- 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.
- 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:

`\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.

```
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 the 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,epno 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,epno 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;

-- Invoke the stored procedure.
CALL regress_record('abc');

-- Delete the stored procedure.
DROP PROCEDURE regress_record;
```

17.5 Syntax

17.5.1 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:

- **DECLARE:** declares variables, types, cursors, and regional stored procedures and functions used in the PL/SQL block.

DECLARE

NOTE

This part is optional if no variable needs 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 part.
- **EXECUTION:** specifies procedure and SQL statements. It is the main part of a program. Mandatory

BEGIN

- **EXCEPTION:** processes errors. Optional

EXCEPTION

- **END**

END;

/

NOTICE

You are not allowed to use consecutive tabs in the PL/SQL block, because they may result in an exception when the parameter **-r** is executed using the **gsq** tool.

Type

PL/SQL blocks are classified into the following types:

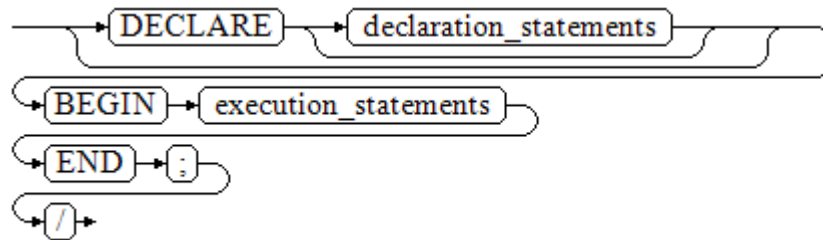
- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see [Figure 17-2](#).
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

17.5.2 Anonymous Block

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 17-2](#) shows the syntax diagrams for an anonymous block.

Figure 17-2 anonymous_block::=

Details about the syntax diagram are 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 (;). 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. 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:
BEGIN
  NULL;
END;
/

-- Print information to the console:
BEGIN
  dbms_output.put_line('hello world!');
END;
/

-- Print variable contents to the console:
DECLARE
  my_var VARCHAR2(30);
BEGIN
  my_var := 'world';
  dbms_output.put_line('hello'||my_var);
END;
/
```

17.5.3 Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

17.6 Basic Statements

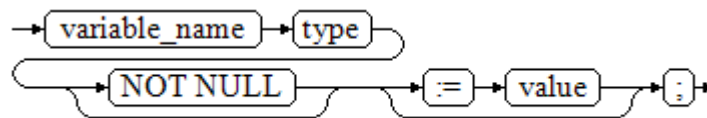
17.6.1 Variable Definition Statement

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 17-3](#).

Figure 17-3 declare_variable::=



The above syntax diagram is explained as follows:

- **variable_name** indicates 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:

```
DECLARE
  emp_id INTEGER := 7788; -- Define a variable and assign a value to it.
BEGIN
  emp_id := 5*7784; -- Assign a value to the variable.
END;
/
```

In addition to the declaration of basic variable types, %TYPE and %ROWTYPE can be used to declare variables related to table columns or table structures.

%TYPE Attribute

%TYPE declares a variable to be of the same data type as a previously declared variable (for example, a column in a table). For example, if you want to define a **my_name** variable whose data type is the same as the data type of the **firstname** column in the **employee** table, you can define the variable as follows:

```
my_name employee.firstname%TYPE
```

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
```

NOTICE

If multiple CNs are used, the %ROWTYPE and %TYPE attributes of temporary tables cannot be declared in a stored procedure, because a temporary table is valid only in the current session and is invisible to other CNs in the compilation phase. In this case, a message is displayed indicating that the temporary table does not exist.

Scope of a Variable

The scope of a variable indicates the accessibility and availability of a 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:

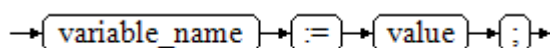
```
DECLARE
  emp_id INTEGER :=7788; -- Define a variable and assign a value to it.
  outer_var INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
  DECLARE
    emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
    inner_var INTEGER :=6688; -- Define a variable and assign a value to it.
  BEGIN
    dbms_output.put_line('inner emp_id ='||emp_id); -- Display the value as 7799.
    dbms_output.put_line('outer_var ='||outer_var); -- Cite variables of an outer block.
  END;
  dbms_output.put_line('outer emp_id ='||emp_id); -- Display the value as 7788.
END;
```

17.6.2 Assignment Statement

Syntax

Figure 17-4 shows the syntax diagram for assigning a value to a variable.

Figure 17-4 assignment_value::=



The above syntax diagram is explained as follows:

- **variable_name** indicates 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**.

Examples

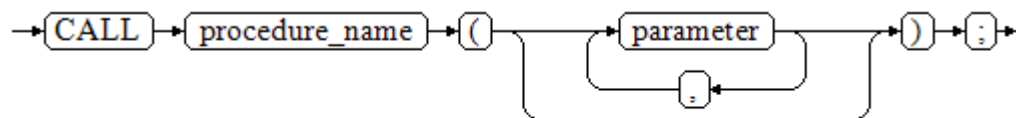
```
DECLARE
  emp_id INTEGER := 7788; --Assignment
BEGIN
  emp_id := 5; --Assignment
  emp_id := 5*7784;
END;
/
```

17.6.3 Call Statement

Syntax

Figure 17-5 shows the syntax diagram for calling a clause.

Figure 17-5 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 the stored procedure proc_staffs:
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:
CREATE OR REPLACE PROCEDURE proc_return
AS
  v_num NUMBER(8,2);
  v_sum INTEGER;
BEGIN
  proc_staffs(30, v_sum, v_num); --Invoke a statement
  dbms_output.put_line(v_sum||'#'||v_num);
RETURN; --Return a statement
```



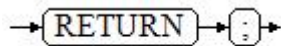
```
END;  
/  
  
-- Invoke a stored procedure proc_return:  
CALL proc_return();  
  
-- Delete a stored procedure:  
DROP PROCEDURE proc_staffs;  
DROP PROCEDURE proc_return;  
  
--Create the function func_return.  
CREATE OR REPLACE FUNCTION func_return returns void  
language plpgsql  
AS $$  
DECLARE  
v_num INTEGER := 1;  
BEGIN  
dbms_output.put_line(v_num);  
RETURN; --Return a statement  
END $$;  
  
-- Invoke the function func_return.  
CALL func_return();  
  
-- Delete the function.  
DROP FUNCTION func_return;
```

17.6.4 Return Statement

Syntax

[Figure 17-6](#) shows the syntax diagram for a return statement.

Figure 17-6 return_clause::=



Details about the syntax diagram are as follows:

This statement returns control from a stored procedure to a caller.

Examples

See [Examples](#) for call statement examples.

Syntax of a Storage Function

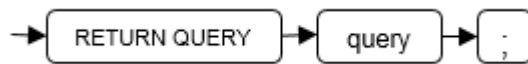
RETURN NEXT and **RETURN QUERY** are used for functions and cannot be used for stored procedures.

When creating a function, specify **SETOF datatype** for the return values.

return_next_clause::=



return_query_clause::=



Details about the syntax diagram are 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 executes a query and adds the query result to the result set of the function. **RETURN QUERY** has a variant **RETURN QUERY EXECUTE**. You can add dynamic queries and add parameters to the queries by using **USING**.

Example of a Memory Function

```
postgres=# CREATE TABLE t1(a int);
postgres=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
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;
postgres=# call fun_for_return_next();
 a
---
 1
10
(2 rows)

-- RETURN QUERY
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;
postgres=# call fun_for_return_query();
 a
---
 1
10
(2 rows)
```

17.7 Dynamic Statements

17.7.1 Executing Dynamic Query Statements

You can perform dynamic queries using **EXECUTE IMMEDIATE** or **OPEN FOR** in GaussDB(DWS). **EXECUTE IMMEDIATE** dynamically executes **SELECT** statements and **OPEN FOR** combines use of cursors. If you need to store query results in a data set, use **OPEN FOR**.

EXECUTE IMMEDIATE

Figure 17-7 shows the syntax diagram.

Figure 17-7 EXECUTE IMMEDIATE dynamic_select_clause::=

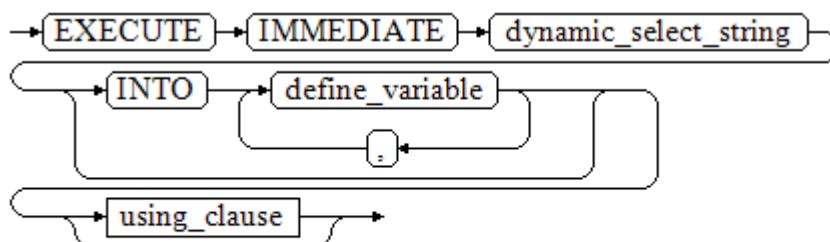
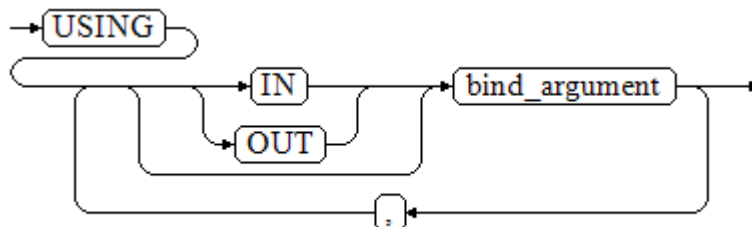


Figure 17-8 shows the syntax diagram for **using_clause**.

Figure 17-8 using_clause::=



The above syntax diagram is explained as follows:

- **define_variable**: specifies variables to store single-line query results.
- **USING IN bind_argument**: specifies where the variable passed to the dynamic SQL value is stored, that is, in the dynamic placeholder of **dynamic_select_string**.
- **USING OUT bind_argument**: specifies where the dynamic SQL returns the value of the variable.

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

```
--Retrieve values from dynamic statements (INTO clause).
DECLARE
  staff_count VARCHAR2(20);
BEGIN
  EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
    INTO staff_count;
  dbms_output.put_line(staff_count);
END;
/

--Pass and retrieve values (the INTO clause is used before the USING clause).
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;
  dbms_output.put_line(first_name || ' ' || salary);
END;
/

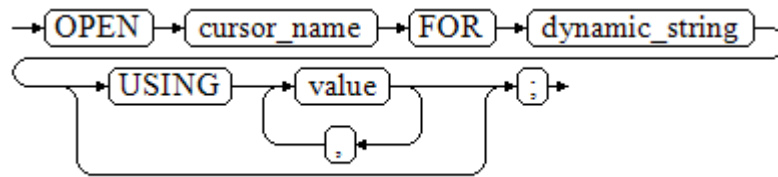
-- Invoke the stored procedure.
CALL dynamic_proc();

-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

Figure 17-9 shows the syntax diagram.

Figure 17-9 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 use of cursors, see [Cursors](#).

Example

```
DECLARE
  name          VARCHAR2(20);
  phone_number  VARCHAR2(20);
  salary        NUMBER(8,2);
  sqlstr        VARCHAR2(1024);

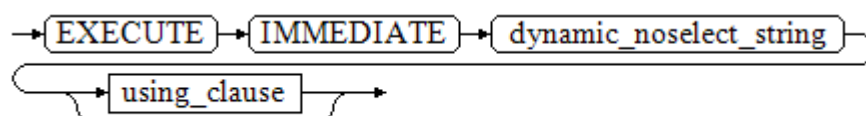
  TYPE app_ref_cur_type IS REF CURSOR; -- Define the cursor type.
  my_cur app_ref_cur_type; -- Define the cursor variable.

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
    dbms_output.put_line(name||'##'||phone_number||'##'||salary);
    FETCH my_cur INTO name, phone_number, salary;
  END LOOP;
  CLOSE my_cur; -- Close the cursor.
END;
/
```

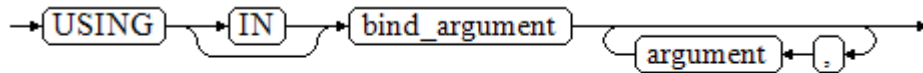
17.7.2 Executing Dynamic Non-query Statements

Syntax

[Figure 17-10](#) shows the syntax diagram.

Figure 17-10 noselect::=

[Figure 17-11](#) shows the syntax diagram for `using_clause`.

Figure 17-11 using_clause::=

The above syntax diagram is explained as follows:

USING IN bind_argument is used to specify the variable that transfers values to dynamic SQL statements. It 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:
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:
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:
SELECT * FROM sections_t1;

--Delete the table.
DROP TABLE sections_t1;
```

17.7.3 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 17-12 shows the syntax diagram.

Figure 17-12 call_procedure::=

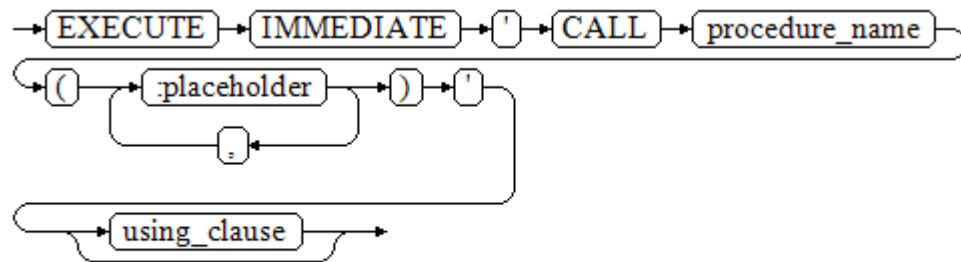
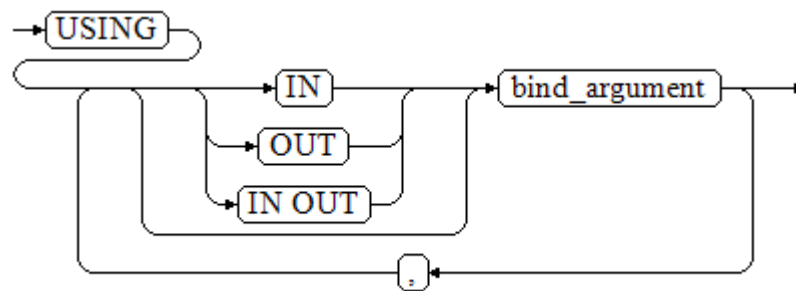


Figure 17-13 shows the syntax diagram for **using_clause**.

Figure 17-13 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 the parameters are the same.
- **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.

Examples

```
--Create the stored procedure proc_add:
CREATE OR REPLACE PROCEDURE proc_add
(
    param1 in INTEGER,
    param2 out INTEGER,
    param3 in INTEGER
)
AS
```

```
BEGIN
  param2:= param1 + param3;
END;
/

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;
  dbms_output.put_line('result is: '||to_char(param2));
END;
/

-- Delete the stored procedure.
DROP PROCEDURE proc_add;
```

17.7.4 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 17-14 shows the syntax diagram.

Figure 17-14 call_anonymous_block::=

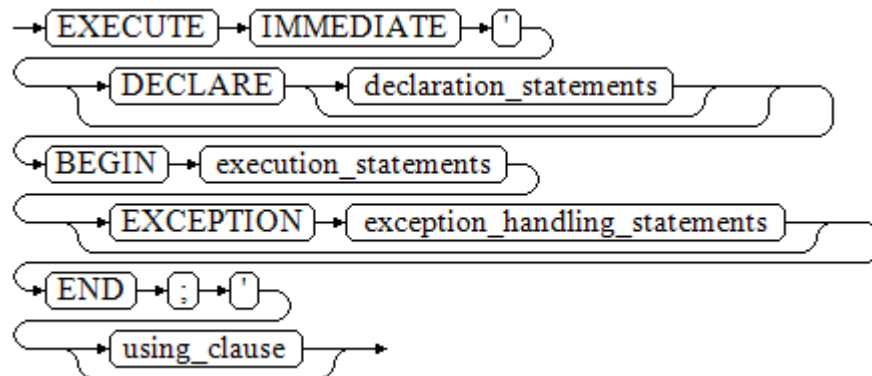
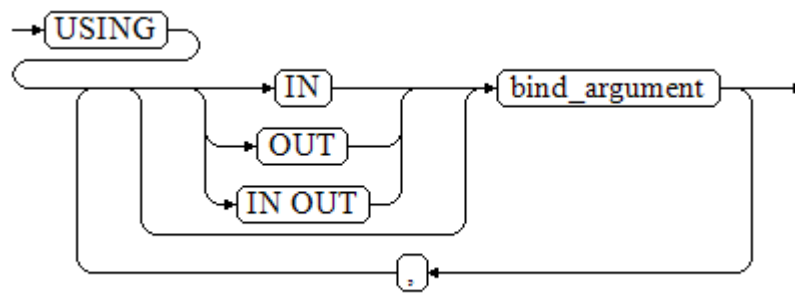


Figure 17-15 shows the syntax diagram for **using_clause**.

Figure 17-15 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 the parameters are the same. The sequences of the parameters corresponding to the placeholders and the USING parameters are the same.
- Currently in GaussDB(DWS), when dynamic statements call anonymous blocks, placeholders cannot be used to pass input and output parameters in an **EXCEPTION** statement.

Examples

```
--Create the stored procedure dynamic_proc.
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;
    dbms_output.put_line(first_name|| ' ' || salary);
END;
/

-- Invoke the stored procedure.
CALL dynamic_proc();

-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

17.8 Control Statements

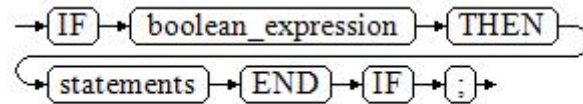
17.8.1 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

GaussDB(DWS) supports five usages of **IF**:

- **IF_THEN**

Figure 17-16 IF_THEN::=



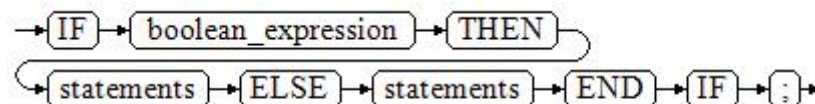
IF_THEN is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

Example

```
IF v_user_id <> 0 THEN
  UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- **IF_THEN_ELSE**

Figure 17-17 IF_THEN_ELSE::=



IF-THEN-ELSE statements add **ELSE** branches and can be executed if the condition is **false**.

Example

```
IF parentid IS NULL OR parentid = ''
THEN
  RETURN;
ELSE
  hp_true_filename(parentid); -- Call the stored procedure.
END IF;
```

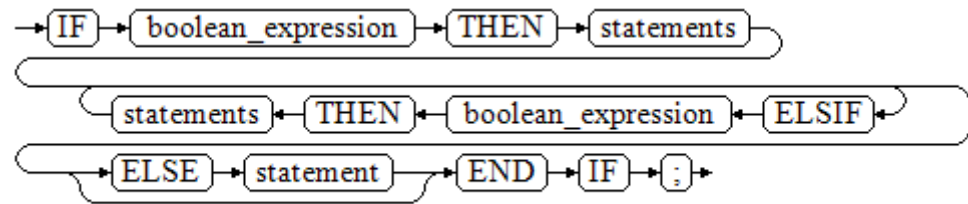
- **IF_THEN_ELSE IF**

IF statements can be nested in the following way:

```
IF sex = 'm' THEN
  pretty_sex := 'man';
ELSE
  IF sex = 'f' THEN
    pretty_sex := 'woman';
  END IF;
END IF;
```

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 to end the parent **IF-ELSE** statement. To set multiple options, use the following form:

- **IF_THEN_ELSEIF_ELSE**

Figure 17-18 IF_THEN_ELSEIF_ELSE::=**Example**

```
IF number_tmp = 0 THEN
    result := 'zero';
ELSIF number_tmp > 0 THEN
    result := 'positive';
ELSIF number_tmp < 0 THEN
    result := 'negative';
ELSE
    result := 'NULL';
END IF;
```

- 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;
/

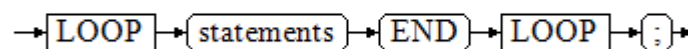
CALL proc_control_structure(3);

-- Delete the stored procedure:
DROP PROCEDURE proc_control_structure;
```

17.8.2 Loop Statements

Simple LOOP Statements

The syntax diagram is as follows.

Figure 17-19 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);
```

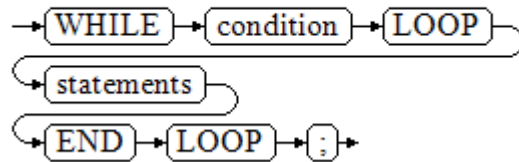
NOTICE

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

WHILE-LOOP Statements

The syntax diagram is as follows.

Figure 17-20 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;
/

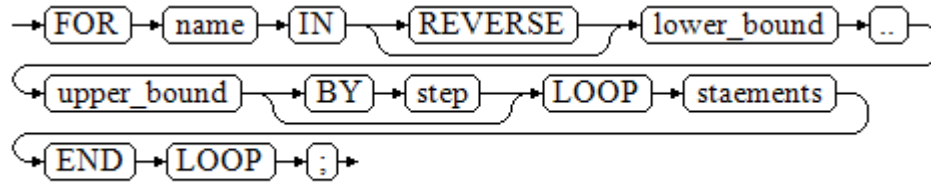
-- Invoke a function:
CALL proc_while_loop(10);

-- Delete the stored procedure and table:
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP (*Integer variable*) Statement

The syntax diagram is as follows.

Figure 17-21 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
    DBMS_OUTPUT.PUT_LINE('It is '||to_char(I) || ' time;');
  END LOOP;
END;
/

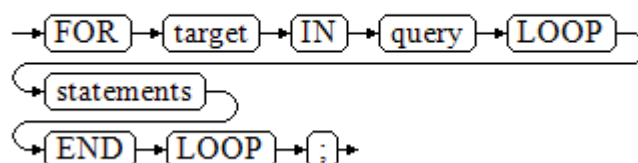
-- Invoke a function:
CALL proc_for_loop();

-- Delete the stored procedure:
DROP PROCEDURE proc_for_loop;
```

FOR_LOOP Query Statements

The syntax diagram is as follows.

Figure 17-22 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
        dbms_output.put_line(record);
    END LOOP;
END;
/

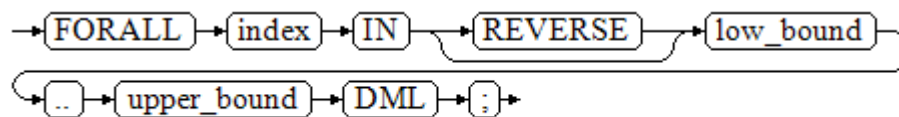
-- Invoke a function.
CALL proc_for_loop_query();

-- Delete the stored procedure.
DROP PROCEDURE proc_for_loop_query;
```

FORALL Batch Query Statements

The syntax diagram is as follows.

Figure 17-23 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 hdfs_t1 (
    title NUMBER(6),
    did VARCHAR2(20),
    data_peroid VARCHAR2(25),
    kind VARCHAR2(25),
    interval VARCHAR2(20),
    time DATE,
    isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);

INSERT INTO hdfs_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 hdfs_t1 set title = title + 100*i;
END;
/

-- Invoke a function:
CALL proc_forall();

-- Query the invocation result of the stored procedure:
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;
```

```
-- Delete the stored procedure and table:  
DROP PROCEDURE proc_forall;  
DROP TABLE hdfs_t1;
```

17.8.3 Branch Statements

Syntax

Figure 17-24 shows the syntax diagram.

Figure 17-24 case_when::=

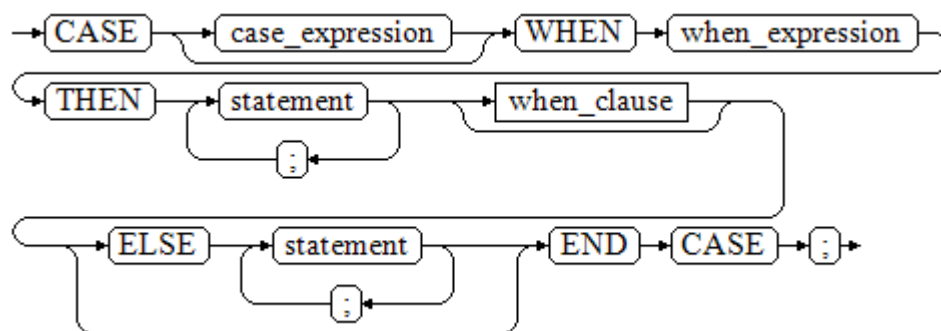
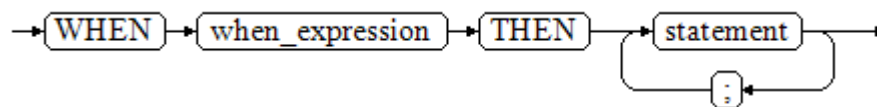


Figure 17-25 shows the syntax diagram for when_clause.

Figure 17-25 when_clause::=



Parameter description:

- **case_expression**: specifies the variable or expression.
- **when_expression**: specifies the constant or conditional expression.
- **statement**: specifies the statement to execute.

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);

-- Delete the stored procedure:
DROP PROCEDURE proc_case_branch;
```

17.8.4 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;
/
```

17.8.5 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:

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.

For 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
-----+-----+-----
   1 | 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 or network fault.

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
$$
BEGIN
    LOOP
        -- First try to update the key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
        -- Not there, so try to insert the key. If someone else inserts the same key concurrently, we could get a
        -- unique-key failure.
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
            -- Do nothing, and loop to try the UPDATE again.
        END;
    END LOOP;
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

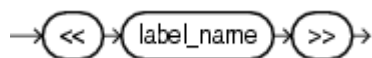
--Delete FUNCTION and TABLE:
DROP FUNCTION merge_db;
DROP TABLE db ;
```

17.8.6 GOTO Statements

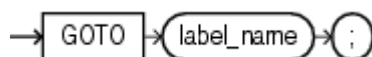
The **GOTO** statement unconditionally transfers the control from the current statement to a labeled statement. The **GOTO** statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the **EXCEPTION** statement to handle issues in special scenarios. To run the **GOTO** statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=



Examples

```
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

The **GOTO** statement has the following constraints:

- The **GOTO** statement does not allow multiple labeled statements even if they are in different blocks.

```
BEGIN
    GOTO pos1;
<<pos1>>
SELECT * FROM ...
<<pos1>>
UPDATE t1 SET ...
END;
```

- The **GOTO** statement cannot transfer control to the **IF**, **CASE**, or **LOOP** statement.

```
BEGIN
    GOTO pos1;
    IF valid THEN
        <<pos1>>
        SELECT * FROM ...
    END IF;
END;
```

- The **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;
```

- The **GOTO** statement cannot transfer control from an outer block to an inner **BEGIN-END** block.

```
BEGIN
    GOTO pos1;
    BEGIN
        <<pos1>>
        UPDATE t1 SET ...
    END;
END;
```

- The **GOTO** statement cannot transfer control from an **EXCEPTION** block to the current **BEGIN-END** block but can transfer to an outer **BEGIN-END** block.

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- If the labeled statement in the **GOTO** statement does not exist, you need to 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;
/
```

17.9 Lock Operations

GaussDB(DWS) 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(DWS) 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.

17.10 Cursors

17.10.1 Overview

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers to context regions. With 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 17-2](#) shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 17-2 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

17.10.2 Explicit Cursor

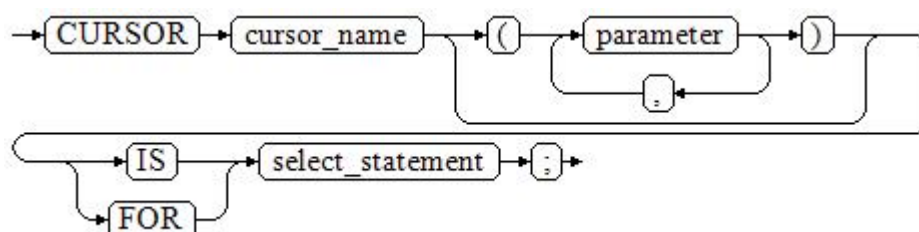
An explicit cursor is used to process query statements, particularly when query results are multiple records.

Procedure

An explicit cursor performs the following six PL/SQL steps to process query statements:

- Step 1 Define a static cursor:** Define a cursor name and its corresponding **SELECT** statement.

Figure 17-26 shows the syntax diagram for defining a static cursor.

Figure 17-26 static_cursor_define::=

Parameter description:

- **cursor_name**: defines a cursor name.
- **parameter**: specifies cursor parameters. Only input parameters are allowed in the following format:
parameter_name datatype
- **select_statement**: specifies a query statement.

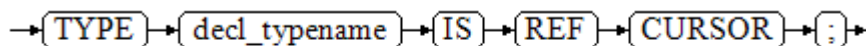
NOTE

The system automatically determines whether the cursor can be used for backward fetches based on the execution plan.

Define a dynamic cursor: Define a **ref** cursor, which means that the cursor can be opened dynamically by a set of static SQL statements. First 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.

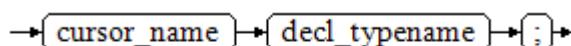
Figure 17-27 and **Figure 17-28** show the syntax diagrams for defining a dynamic cursor.

Figure 17-27 cursor_type_name::=



GaussDB(DWS) 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.

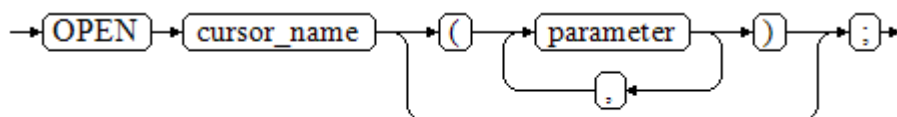
Figure 17-28 dynamic_cursor_define::=



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 row corresponding to the cursor result set in the database table.

Figure 17-29 shows the syntax diagram for opening a static cursor.

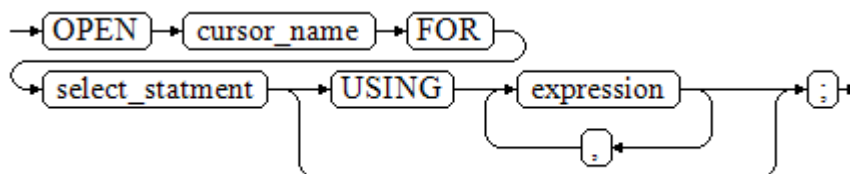
Figure 17-29 open_static_cursor::=



Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 17-30 shows the syntax diagrams for opening a dynamic cursor.

Figure 17-30 open_dynamic_cursor::=

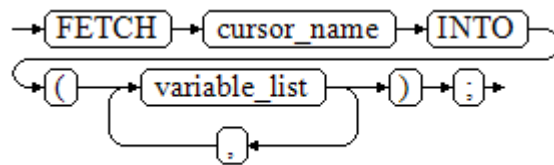


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

Step 3 Fetch cursor data: Retrieve data rows in the result set and place them in specified output variables.

Figure 17-31 shows the syntax diagrams for fetching cursor data.

Figure 17-31 fetch_cursor::=



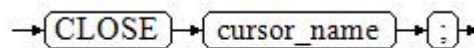
Step 4 Process the record.

Step 5 Continue to process until the active set has no record.

Step 6 Close the cursor: When fetching and finishing the data in the cursor result set, close the cursor immediately to release system resources used by the cursor and invalidate the workspace of the cursor so that the **FETCH** statement cannot be used to fetch data any more. A closed cursor can be reopened by an **OPEN** statement.

Figure 17-32 shows the syntax diagram for closing a cursor.

Figure 17-32 close_cursor::=



----End

Attribute

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` attribute: returns **TRUE** if the last fetch returns a row.
- `%NOTFOUND` attribute: works opposite to the `%FOUND` attribute.
- `%ISOPEN` attribute: returns **TRUE** if the cursor has been opened.
- `%ROWCOUNT` attribute: returns the number of records fetched from the cursor.

Example

```
-- 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;
    DBMS_OUTPUT.PUT_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;
    DBMS_OUTPUT.PUT_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;
    DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
  END LOOP;
  CLOSE C3;
END;
/

CALL cursor_proc1();

DROP PROCEDURE cursor_proc1;
-- 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;
/

CALL cursor_proc2();

-- Delete the stored procedure:
DROP PROCEDURE cursor_proc2;
DROP TABLE hr.staffs_t1;
-- 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;  
    DBMS_OUTPUT.PUT_LINE(C1%ROWCOUNT);  
    EXIT WHEN C1%NOTFOUND;  
END LOOP;  
END;  
/  
  
-- Delete the stored procedure.  
DROP PROCEDURE proc_sys_ref;
```

17.10.3 Implicit Cursor

Implicit cursors are automatically set by the system for non-query statements such as modify or delete operations, along with their workspace. Implicit cursors are named SQL, which is defined by the system.

Overview

Implicit cursor operations, such as definition, open, value-grant, and close operations, are automatically performed by the system and do not need users to process. Users can use only attributes related to implicit cursors to complete operations. In workspace of implicit cursors, the data of the latest SQL statement is stored and is not related to explicit cursors defined by users.

Format call: SQL%

NOTE

INSERT, **UPDATE**, **DELETE**, and **SELECT** statements do not need defined cursors.

Attribute

An implicit cursor has the following attributes:

- **SQL%FOUND** attribute: returns **TRUE** if the last fetch returns a row.
- **SQL%NOTFOUND** attribute: works opposite to the **%FOUND** attribute.
- **SQL%ROWCOUNT** attribute: returns the number of records fetched from the cursor.
- **SQL%ISOPEN** attribute: always takes **FALSE**. Close implicit cursors immediately after an SQL statement is run.

Example

```
-- Delete all employees in a department from the EMP table. If the department has no member, delete the  
department from the DEPT table.  
CREATE TABLE hr.staffs_t1AS TABLE hr.staffs;  
CREATE TABLE hr.sections_t1AS 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();

-- Delete the stored procedure and the temporary table:
DROP PROCEDURE proc_cursor3;
DROP TABLE hr.staffs_t1;
DROP TABLE hr.sections_t1;
```

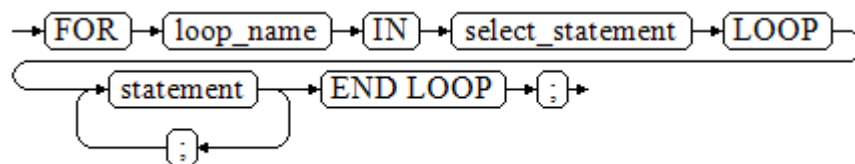
17.10.4 Cursor Loop

Use of cursors in WHILE and LOOP statements is called a cursor loop. Generally, OPEN, FETCH, and CLOSE statements are involved in this kind of loop. The following describes a loop that simplifies a cursor loop without the need for these operations. This kind of loop is applicable to a static cursor loop, without executing four steps about a static cursor.

Syntax

Figure 17-33 shows the syntax diagram of the **FOR AS** loop.

Figure 17-33 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 **select_statement** query result. 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(DWS). Transactions and the anonymous block do not support multiple cursors accesses at the same time.

Example

```
BEGIN
FOR ROW_TRANS IN
  SELECT first_name FROM hr.staffs
LOOP
  DBMS_OUTPUT.PUT_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 value of C1%FOUND and C2%FOUND is FALSE.
FETCH C2 INTO PI_B; ---- The value of C1%FOUND and C2%FOUND is TRUE.
-- Determine the cursor status:
IF C1%FOUND THEN
IF C2%FOUND THEN
DBMS_OUTPUT.PUT_LINE('Dual cursor share parameter.');
```

```
END IF;
CLOSE C1;-- Close the cursor.
CLOSE C2;
END;
/
```

```
-- Delete the temporary table:
DROP TABLE integerTable1;
DROP TABLE integerTable2;
```

17.11 Advanced Packages

17.11.1 DBMS_LOB

Related Interfaces

[Table 17-3](#) provides all interfaces supported by the **DBMS_LOB** package.

Table 17-3 DBMS_LOB

API	Description
DBMS_LOB.GETLENGTH	Obtains and returns the specified length of a LOB object.
DBMS_LOB.OPEN	Opens a LOB and returns a LOB descriptor.
DBMS_LOB.READ	Loads a part of LOB contents to BUFFER area according to the specified length and initial position offset.
DBMS_LOB.WRITE	Copies contents in BUFFER area to LOB according to the specified length and initial position offset.
DBMS_LOB.WRITEAPPEND	Copies contents in BUFFER area to the end part of LOB according to the specified length.

API	Description
DBMS_LOB.COPY	Copies contents in BLOB to another BLOB according to the specified length and initial position offset.
DBMS_LOB.ERASE	Deletes contents in BLOB according to the specified length and initial position offset.
DBMS_LOB.CLOSE	Closes a LOB descriptor.
DBMS_LOB.INSTR	Returns the position of the Nth occurrence of a character string in LOB.
DBMS_LOB.COMPARE	Compares two LOBs or a certain part of two LOBs.
DBMS_LOB.SUBSTR	Reads the substring of a LOB and returns the number of read bytes or the number of characters.
DBMS_LOB.TRIM	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.
DBMS_LOB.CREATETEMPORARY	Creates a temporary BLOB or CLOB.
DBMS_LOB.APPEND	Adds the content of a LOB to another LOB.

- **DBMS_LOB.GETLENGTH**

Specifies the length of a LOB type object obtained and returned by the stored procedure **GETLENGTH**.

The function prototype of **DBMS_LOB.GETLENGTH** is:

```
DBMS_LOB.GETLENGTH (  
lob_loc IN BLOB)  
RETURN INTEGER;  
  
DBMS_LOB.GETLENGTH (  
lob_loc IN CLOB)  
RETURN INTEGER;
```

Table 17-4 DBMS_LOB.GETLENGTH interface parameters

Parameter	Description
lob_loc	LOB type object whose length is to be obtained

- **DBMS_LOB.OPEN**

A stored procedure opens a LOB and returns a LOB descriptor. This process is used only for compatibility.

The function prototype of **DBMS_LOB.OPEN** is:

```
DBMS_LOB.LOB (  
lob_loc INOUT BLOB,  
open_mode IN BINARY_INTEGER);
```

```
DBMS_LOB.LOB (  
lob_loc INOUT CLOB,  
open_mode IN BINARY_INTEGER);
```

Table 17-5 DBMS_LOB.OPEN interface parameters

Parameter	Description
lob_loc	BLOB or CLOB descriptor that is opened
open_mode IN BINARY_INTEG ER	Open mode (currently, DBMS_LOB.LOB_READWRITE is supported)

- DBMS_LOB.READ

The stored procedure **READ** loads a part of LOB contents to BUFFER according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.READ (  
lob_loc IN BLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer OUT RAW);  
  
DBMS_LOB.READ (  
lob_loc IN CLOB,  
amount IN OUT INTEGER,  
offset IN INTEGER,  
buffer OUT VARCHAR2);
```

Table 17-6 DBMS_LOB.READ interface parameters

Parameter	Description
lob_loc	LOB type object to be loaded
amount	Load data length NOTE If the read length is negative, the error message "ERROR: argument 2 is null, invalid, or out of range." is displayed.
offset	Indicates where to start reading the LOB contents, that is, the offset bytes to initial position of LOB contents.
buffer	Target buffer to store the loaded LOB contents

- DBMS_LOB.WRITE

The stored procedure **WRITE** copies contents in BUFFER to LOB variables according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.WRITE** is:

```
DBMS_LOB.WRITE (  
lob_loc IN OUT BLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer IN RAW);
```

```
DBMS_LOB.WRITE (  
lob_loc IN OUT CLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer IN VARCHAR2);
```

Table 17-7 DBMS_LOB.WRITE interface parameters

Parameter	Description
lob_loc	LOB type object to be written
amount	Write data length NOTE If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
offset	Indicates where to start writing the LOB contents, that is, the offset bytes to initial position of LOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of LOB type contents, an error is reported.
buffer	Content to be written

- DBMS_LOB.WRITEAPPEND

The stored procedure **WRITEAPPEND** copies contents in BUFFER to the end part of LOB according to the specified length.

The function prototype of **DBMS_LOB.WRITEAPPEND** is:

```
DBMS_LOB.WRITEAPPEND (  
lob_loc IN OUT BLOB,  
amount IN INTEGER,  
buffer IN RAW);  
  
DBMS_LOB.WRITEAPPEND (  
lob_loc IN OUT CLOB,  
amount IN INTEGER,  
buffer IN VARCHAR2);
```

Table 17-8 DBMS_LOB.WRITEAPPEND interface parameters

Parameter	Description
lob_loc	LOB type object to be written
amount	Write data length NOTE If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
buffer	Content to be written

- DBMS_LOB.COPY

The stored procedure **COPY** copies contents in BLOB to another BLOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.COPY** is:

```
DBMS_LOB.COPY (  
dest_lob    IN OUT   BLOB,  
src_lob     IN       BLOB,  
amount      IN       INTEGER,  
dest_offset IN       INTEGER DEFAULT 1,  
src_offset  IN       INTEGER DEFAULT 1);
```

Table 17-9 DBMS_LOB.COPY interface parameters

Parameter	Description
dest_lob	BLOB type object to be pasted
src_lob	BLOB type object to be copied
amount	Length of the copied data NOTE If the copied data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
dest_offset	Indicates where to start pasting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
src_offset	Indicates where to start copying the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the length of source BLOB, an error is reported.

- DBMS_LOB.ERASE

The stored procedure **ERASE** deletes contents in BLOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.ERASE** is:

```
DBMS_LOB.ERASE (  
lob_loc     IN OUT   BLOB,  
amount      IN OUT   INTEGER,  
offset      IN       INTEGER DEFAULT 1);
```

Table 17-10 DBMS_LOB.ERASE interface parameters

Parameter	Description
lob_loc	BLOB type object whose contents are to be deleted
amount	Length of contents to be deleted NOTE If the deleted data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.

Parameter	Description
offset	Indicates where to start deleting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.

- **DBMS_LOB.CLOSE**

The procedure **CLOSE** disables the enabled contents of LOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.CLOSE** is:

```
DBMS_LOB.CLOSE(  
src_lob      IN      BLOB);
```

```
DBMS_LOB.CLOSE (  
src_lob      IN      CLOB);
```

Table 17-11 DBMS_LOB.CLOSE interface parameters

Parameter	Description
src_loc	LOB type object to be disabled

- **DBMS_LOB.INSTR**

This function returns the Nth occurrence position in LOB. If invalid values are entered, **NULL** is returned. The invalid values include offset < 1 or offset > LOBMAXSIZE, nth < 1, and nth > LOBMAXSIZE.

The function prototype of **DBMS_LOB.INSTR** is:

```
DBMS_LOB.INSTR (  
lob_loc      IN      BLOB,  
pattern      IN      RAW,  
offset       IN      INTEGER := 1,  
nth          IN      INTEGER := 1)  
RETURN INTEGER;
```

```
DBMS_LOB.INSTR (  
lob_loc      IN      CLOB,  
pattern      IN      VARCHAR2 ,  
offset       IN      INTEGER := 1,  
nth          IN      INTEGER := 1)  
RETURN INTEGER;
```

Table 17-12 DBMS_LOB.INSTR interface parameters

Parameter	Description
lob_loc	LOB descriptor to be searched for
pattern	Matched pattern. It is RAW for BLOB and TEXT for CLOB.
offset	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.

Parameter	Description
nth	Number of pattern matching times. The minimum value is 1.

- DBMS_LOB.COMPARE

This function compares two LOBs or a certain part of two LOBs.

- If the two parts are equal, **0** is returned. Otherwise, a non-zero value is returned.
- If the first CLOB is smaller than the second, **-1** is returned. If the first CLOB is larger than the second, **1** is returned.
- If any of the **amount**, **offset_1**, and **offset_2** parameters is invalid, **NULL** is returned. The valid offset range is 1 to LOBMAXSIZE.

The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.COMPARE (  
lob_1   IN   BLOB,  
lob_2   IN   BLOB,  
amount  IN   INTEGER := DBMS_LOB.LOBMAXSIZE,  
offset_1 IN   INTEGER := 1,  
offset_2 IN   INTEGER := 1)  
RETURN INTEGER;  
  
DBMS_LOB.COMPARE (  
lob_1   IN   CLOB,  
lob_2   IN   CLOB,  
amount  IN   INTEGER := DBMS_LOB.LOBMAXSIZE,  
offset_1 IN   INTEGER := 1,  
offset_2 IN   INTEGER := 1)  
RETURN INTEGER;
```

Table 17-13 DBMS_LOB.COMPARE interface parameters

Parameter	Description
lob_1	First LOB descriptor to be compared
lob_2	Second LOB descriptor to be compared
amount	Number of characters or bytes to be compared. The maximum value is DBMS_LOB.LOBMAXSIZE.
offset_1	Offset of the first LOB descriptor. The initial position is 1.
offset_2	Offset of the second LOB descriptor. The initial position is 1.

- DBMS_LOB.SUBSTR

This function reads the substring of a LOB and returns the number of read bytes or the number of characters. If amount > 1, amount < 32767, offset < 1, or offset > LOBMAXSIZE, **NULL** is returned.

The function prototype of **DBMS_LOB.SUBSTR** is:

```
DBMS_LOB.SUBSTR (  
lob_loc IN   BLOB,  
amount  IN   INTEGER := 32767,
```

```
offset IN INTEGER := 1)
RETURN RAW;

DBMS_LOB.SUBSTR (
lob_loc IN CLOB,
amount IN INTEGER := 32767,
offset IN INTEGER := 1)
RETURN VARCHAR2;
```

Table 17-14 DBMS_LOB.SUBSTR interface parameters

Parameter	Description
lob_loc	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.
offset	Number of bytes or characters to be read.
buffer	Number of characters or bytes offset from the start position.

- DBMS_LOB.TRIM

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 LOB, an exception occurs.

The function prototype of **DBMS_LOB.TRIM** is:

```
DBMS_LOB.TRIM (
lob_loc IN OUT BLOB,
newlen IN INTEGER);

DBMS_LOB.TRIM (
lob_loc IN OUT CLOB,
newlen IN INTEGER);
```

Table 17-15 DBMS_LOB.TRIM interface parameters

Parameter	Description
lob_loc	BLOB type object to be read
newlen	After truncation, the new LOB length for BLOB is in the unit of byte and that for CLOB is in the unit of character.

- DBMS_LOB.CREATETEMPORARY

This stored procedure creates a temporary BLOB or CLOB and is used only for syntax compatibility.

The function prototype of **DBMS_LOB.CREATETEMPORARY** is:

```
DBMS_LOB.CREATETEMPORARY (
lob_loc IN OUT BLOB,
cache IN BOOLEAN,
dur IN INTEGER);
```

```
DBMS_LOB.CREATETEMPORARY (  
lob_loc IN OUT CLOB,  
cache IN BOOLEAN,  
dur IN INTEGER);
```

Table 17-16 DBMS_LOB.CREATETEMPORARY interface parameters

Parameter	Description
lob_loc	LOB descriptor
cache	This parameter is used only for syntax compatibility.
dur	This parameter is used only for syntax compatibility.

- DBMS_LOB.APPEND

The stored procedure **READ** loads a part of BLOB contents to BUFFER according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.APPEND** is:

```
DBMS_LOB.APPEND (  
dest_lob IN OUT BLOB,  
src_lob IN BLOB);  
  
DBMS_LOB.APPEND (  
dest_lob IN OUT CLOB,  
src_lob IN CLOB);
```

Table 17-17 DBMS_LOB.APPEND interface parameters

Parameter	Description
dest_lob	LOB descriptor to be written
src_lob	LOB descriptor to be read

Examples

```
-- Obtain the length of the character string.  
SELECT DBMS_LOB.GETLENGTH('12345678');  
  
DECLARE  
myraw RAW(100);  
amount INTEGER :=2;  
buffer INTEGER :=1;  
begin  
DBMS_LOB.READ('123456789012345',amount,buffer,myraw);  
dbms_output.put_line(myraw);  
end;  
/  
  
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 := utl_raw.cast_to_raw(str);
amount := utl_raw.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;

DBMS_LOB.WRITE(dest, amount, 1, source);
DBMS_LOB.WRITEAPPEND(dest, amount, source);

DBMS_LOB.ERASE(dest, a, 1);
DBMS_OUTPUT.PUT_LINE(a);
DBMS_LOB.COPY(copyto, dest, amount, 10, 1);
DBMS_LOB.CLOSE(dest);
RETURN;
END;
/

--Delete the table.
DROP TABLE blob_Table;
DROP TABLE blob_Table_bak;
```

17.11.2 DBMS_RANDOM

Related Interfaces

[Table 17-18](#) provides all interfaces supported by the **DBMS_RANDOM** package.

Table 17-18 DBMS_RANDOM interface parameters

API	Description
DBMS_RANDOM.SEED	Sets a seed for a random number.
DBMS_RANDOM.VALUE	Generates a random number between a specified low and a specified high.

- DBMS_RANDOM.SEED

The stored procedure SEED is used to set a seed for a random number. The DBMS_RANDOM.SEED function prototype is:

```
DBMS_RANDOM.SEED (seed IN INTEGER);
```

Table 17-19 DBMS_RANDOM.SEED interface parameters

Parameter	Description
seed	Generates a seed for a random number.

- DBMS_RANDOM.VALUE

The stored procedure VALUE generates a random number between a specified low and a specified high. The DBMS_RANDOM.VALUE function prototype is:

```
DBMS_RANDOM.VALUE(  
low IN NUMBER,  
high IN NUMBER)  
RETURN NUMBER;
```

Table 17-20 DBMS_RANDOM.VALUE interface parameters

Parameter	Description
low	Sets the low bound for a random number. The generated random number is greater than or equal to the low.
high	Sets the high bound for a random number. The generated random number is less than the high.

 NOTE

The only requirement is that the parameter type is **NUMERIC** regardless of the right and left bound values.

Examples

```
-- Generate a random number between 0 and 1:  
SELECT DBMS_RANDOM.VALUE(0,1);  
  
-- Add the low and high parameters to an integer within the specified range and intercept smaller values  
from the result. (The maximum value cannot be a possible value.) Therefore, use the following code for an  
integer between 0 and 99:  
SELECT TRUNC(DBMS_RANDOM.VALUE(0,100));
```

17.11.3 DBMS_OUTPUT

Related Interfaces

[Table 17-21](#) provides all interfaces supported by the **DBMS_OUTPUT** package.

Table 17-21 DBMS_OUTPUT

API	Description
DBMS_OUTPUT.PUT_LINE	Outputs the specified text. The text length cannot exceed 32,767 bytes.
DBMS_OUTPUT.PUT	Outputs the specified text to the front of the specified text without adding a line break. The text length cannot exceed 32,767 bytes.

API	Description
DBMS_OUTPUT.ENABLE	Sets the buffer area size. If this interface is not specified, the maximum buffer size is 20,000 bytes and the minimum buffer size is 2000 bytes. If the specified buffer size is less than 2000 bytes, the default minimum buffer size is applied.

- DBMS_OUTPUT.PUT_LINE

The PUT_LINE procedure writes a row of text carrying a line end symbol in the buffer. The DBMS_OUTPUT.PUT_LINE function prototype is:

```
DBMS_OUTPUT.PUT_LINE (  
item IN VARCHAR2);
```

Table 17-22 DBMS_OUTPUT.PUT_LINE interface parameters

Parameter	Description
item	Specifies the text that was written to the buffer.

- DBMS_OUTPUT.PUT

The stored procedure **PUT** outputs the specified text to the front of the specified text without adding a linefeed. The DBMS_OUTPUT.PUT function prototype is:

```
DBMS_OUTPUT.PUT (  
item IN VARCHAR2);
```

Table 17-23 DBMS_OUTPUT.PUT interface parameters

Parameter	Description
item	Specifies the text that was written to the specified text.

- DBMS_OUTPUT.ENABLE

The stored procedure **ENABLE** sets the output buffer size. If the size is not specified, it contains a maximum of 20,000 bytes. The DBMS_OUTPUT.ENABLE function prototype is:

```
DBMS_OUTPUT.ENABLE (  
buf IN INTEGER);
```

Table 17-24 DBMS_OUTPUT.ENABLE interface parameters

Parameter	Description
buf	Sets the buffer area size.

Examples

```
BEGIN
  DBMS_OUTPUT.ENABLE(50);
  DBMS_OUTPUT.PUT ('hello, ');
  DBMS_OUTPUT.PUT_LINE('database!');-- Displaying "hello, database!"
END;
/
```

17.11.4 UTL_RAW

Related Interfaces

[Table 17-25](#) provides all interfaces supported by the **UTL_RAW** package.

Table 17-25 UTL_RAW

API	Description
UTL_RAW.CAST_FROM_BINARY_INTEGER	Converts an INTEGER type value to a binary representation (RAW type).
UTL_RAW.CAST_TO_BINARY_INTEGER	Converts a binary representation (RAW type) to an INTEGER type value.
UTL_RAW.LENGTH	Obtains the length of the RAW type object.
UTL_RAW.CAST_TO_RAW	Converts a VARCHAR2 type value to a binary expression (RAW type).

NOTICE

The external representation of the **RAW** type data is hexadecimal and its internal storage form is binary. For example, the representation of the **RAW** type data **11001011** is 'CB'. The input of the actual type conversion is 'CB'.

- **UTL_RAW.CAST_FROM_BINARY_INTEGER**
The stored procedure **CAST_FROM_BINARY_INTEGER** converts an **INTEGER** type value to a binary representation (**RAW** type).

The **UTL_RAW.CAST_FROM_BINARY_INTEGER** function prototype is:

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
  n          IN INTEGER,
  endianness IN INTEGER)
RETURN RAW;
```

Table 17-26 UTL_RAW.CAST_FROM_BINARY_INTEGER interface parameters

Parameter	Description
n	Specifies the INTEGER type value to be converted to the RAW type.

Parameter	Description
endianess	Specifies the INTEGER type value 1 or 2 of the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates LITTLE-ENDIAN .)

- UTL_RAW.CAST_TO_BINARY_INTEGER

The stored procedure CAST_TO_BINARY_INTEGER converts an INTEGER type value in a binary representation (RAW type) to the INTEGER type.

The UTL_RAW.CAST_TO_BINARY_INTEGER function prototype is:

```
UTL_RAW.CAST_TO_BINARY_INTEGER (  
r      IN RAW,  
endianess IN INTEGER)  
RETURN BINARY_INTEGER;
```

Table 17-27 UTL_RAW.CAST_TO_BINARY_INTEGER interface parameters

Parameter	Description
r	Specifies an INTEGER type value in a binary representation (RAW type).
endianess	Specifies the INTEGER type value 1 or 2 of the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates LITTLE-ENDIAN .)

- UTL_RAW.LENGTH

The stored procedure LENGTH returns the length of a RAW type object.

The UTL_RAW.LENGTH function prototype is:

```
UTL_RAW.LENGTH(  
r      IN RAW)  
RETURN INTEGER;
```

Table 17-28 UTL_RAW.LENGTH interface parameters

Parameter	Description
r	Specifies a RAW type object.

- UTL_RAW.CAST_TO_RAW

The stored procedure CAST_TO_RAW converts a VARCHAR2 type object to the RAW type.

The UTL_RAW.CAST_TO_RAW function prototype is:

```
UTL_RAW.CAST_TO_RAW(  
c      IN VARCHAR2)  
RETURN RAW;
```


Table 17-29 UTL_RAW.CAST_TO_RAW interface parameters

Parameter	Description
c	Specifies a VARCHAR2 type object to be converted.

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 := utl_raw.cast_to_raw(str);--Convert the type.
amount := utl_raw.length(source);--Obtain the length.
dbms_output.put_line(amount);
END;
/

-- Invoke the stored procedure.
CALL proc_raw();

-- Delete the stored procedure.
DROP PROCEDURE proc_raw;
```

17.11.5 DBMS_JOB

Related Interfaces

Table 17-30 lists all interfaces supported by the **DBMS_JOB** package.

Table 17-30 DBMS_JOB

API	Description
DBMS_JOB.SUBMIT	Submits a job to the job queue. The job number is automatically generated by the system.
DBMS_JOB.ISUBMIT	Submits a job to the job queue. The job number is specified by the user.
DBMS_JOB.REMOVE	Removes a job from the job queue by job number.
DBMS_JOB.BROKEN	Disables or enables job execution.
DBMS_JOB.CHANGE	Modifies user-definable attributes of a job, including the job content, next-execution time, and execution interval.
DBMS_JOB.WHAT	Modifies the job description for a specified job.
DBMS_JOB.NEXT_DATE	Modifies the next-execution time attribute of a job.

API	Description
DBMS_JOB.INTERVAL	Modifies the execution interval attribute of a job.

- DBMS_JOB.SUBMIT

The stored procedure **SUBMIT** submits a job provided by the system.

A prototype of the DBMS_JOB.SUBMIT function is as follows:

```
DBMS_JOB.SUBMIT(  
  what      IN  TEXT,  
  next_date IN  TIMESTAMP DEFAULT sysdate,  
  job_interval IN TEXT DEFAULT 'null',  
  job       OUT INTEGER);
```

 NOTE

When a job is created (using DBMS_JOB), the system binds the current database and the user name to the job by default. This function can be invoked by using **call** or **select**. If you invoke this function by using **select**, there is no need to specify output parameters. To invoke this function within a stored procedure, use **perform**.

Table 17-31 DBMS_JOB.SUBMIT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	No	SQL statement to be executed. One or multiple DMLs, anonymous blocks, and SQL statements that invoke stored procedures, or all three combined are supported.
next_date	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 submitted.
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 blank 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 number. The value ranges from 1 to 32767. When dbms.submit is invoked using select , this parameter can be skipped.

For example:

```
select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');  
  
select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1.0/24');  
  
CALL DBMS_JOB.SUBMIT('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);
```

- **DBMS_JOB.ISUBMIT**

ISUBMIT has the same syntax function as **SUBMIT**, but the first parameter of **ISUBMIT** is an input parameter, that is, a specified job number. In contrast, that last parameter of **SUBMIT** is an output parameter, indicating the job number automatically generated by the system.

For example:

```
CALL dbms_job.isubmit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
```

- **DBMS_JOB.REMOVE**

The stored procedure **REMOVE** deletes a specified job.

A prototype of the DBMS_JOB.REMOVE function is as follows:

```
REMOVE(job IN INTEGER);
```

Table 17-32 DBMS_JOB.REMOVE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

For example:

```
CALL dbms_job.remove(101);
```

- **DBMS_JOB.BROKEN**

The stored procedure **BROKEN** sets the broken flag of a job.

A prototype of the DBMS_JOB.BROKEN function is as follows:

```
DBMS_JOB.BROKEN(  
job IN INTEGER,  
broken IN BOOLEAN,  
next_date IN TIMESTAMP DEFAULT sysdate);
```

Table 17-33 DBMS_JOB.BROKEN interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
broken	Boolean	IN	No	Specifies the status flag, true for broken and false for not broken. Setting this parameter to true or false updates the current job. If the parameter is left blank, the job status remains unchanged.
next_date	timestamp	IN	Yes	Specifies the next execution time. The default is the current system time. If broken is set to true , next_date is updated to '4000-1-1'. If broken is false and next_date is not empty, next_date is updated for the job. If next_date is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

For example:

```
CALL dbms_job.broken(101, true);  
CALL dbms_job.broken(101, false, sysdate);
```

- **DBMS_JOB.CHANGE**

The stored procedure **CHANGE** modifies user-definable attributes of a job, including the job content, next-execution time, and execution interval.

A prototype of the DBMS_JOB.CHANGE function is as follows:

```
DBMS_JOB.CHANGE(  
  job      IN  INTEGER,  
  what     IN  TEXT,  
  next_date IN  TIMESTAMP,  
  interval IN  TEXT);
```

Table 17-34 DBMS_JOB.CHANGE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left blank, 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 blank, 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.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank, the system does not update the interval parameter for the specified job. Otherwise, the system updates the 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.

For example:

```
CALL dbms_job.change(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');  
CALL dbms_job.change(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
```

- **DBMS_JOB.WHAT**

The stored procedure **WHAT** modifies the procedures to be executed by a specified job.

A prototype of the DBMS_JOB.WHAT function is as follows:

```
DBMS_JOB.WHAT(  
  job      IN   INTEGER,  
  what     IN   TEXT);
```

Table 17-35 DBMS_JOB.WHAT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
what	text	IN	No	Specifies the name of the stored procedure or SQL statement block that is executed.

 NOTE

- If the value specified by the **what** 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 **what** parameter is a simple statement such as insert and update, a schema name must be added in front of the table name.

For example:

```
CALL dbms_job.what(101, 'call userproc();');  
CALL dbms_job.what(101, 'insert into tbl_a values(sysdate);');
```

- **DBMS_JOB.NEXT_DATE**

The stored procedure **NEXT_DATE** modifies the next-execution time attribute of a job.

A prototype of the DBMS_JOB.NEXT_DATE function is as follows:

```
DBMS_JOB.NEXT_DATE(  
  job      IN  INTEGER,  
  next_date IN  TIMESTAMP);
```

Table 17-36 DBMS_JOB.NEXT_DATE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
next_date	timestamp	IN	No	Specifies the next execution time.

 NOTE

If the specified **next_date** value is earlier than the current date, the job is executed once immediately.

For example:

```
CALL dbms_job.next_date(101, sysdate);
```

- DBMS_JOB.INTERVAL

The stored procedure **INTERVAL** modifies the execution interval attribute of a job.

A prototype of the DBMS_JOB.INTERVAL function is as follows:

```
DBMS_JOB.INTERVAL(  
  job      IN  INTEGER,  
  interval IN  TEXT);
```

Table 17-37 DBMS_JOB.INTERVAL interface parameters

Parameter	Type	Input / Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank or set to null , the job will be executed only once, and the job status will change to 'd' afterward. interval must be a valid time or interval type.

For example:

```
CALL dbms_job.interval(101, 'sysdate + 1.0/1440');
```

 NOTE

For a job that is currently running (that is, **job_status** is 'r'), it is not allowed to use **remove**, **change**, **next_date**, **what**, or **interval** to delete or modify job parameters.

Constraints

- After a new job is created, this job belongs to the current coordinator only, that is, this job can be scheduled and executed only on the current coordinator. Other coordinators will not schedule or execute this job. All coordinators can query, modify, and delete jobs created on other CNs.
- Create, update, and delete jobs only using the procedures provided by the DBMS_JOB package. These procedures synchronize job information between different CNs and associate primary keys between the pg_job and pg_job_proc tables. If you use DML statements to add, delete, or modify records in the pg_job table, job information will become inconsistent between CNs and system tables may fail to be associated, compromising internal job management.
- Each job created by a user is bound to a CN. If the CN fails while a job is being executed, the job status cannot be updated in real time and will stay at

'r'. The job status will be updated to 's' only after the CN recovers. When a CN fails, all jobs on this CN cannot be scheduled or executed until the CN is restored manually, or deleted and then replaced.

4. For each job, the hosting CN updates the real-time job information (including the job status, last execution start time, last execution end time, next execution start time, the number of execution failures [if any]) to the **pg_job** table, and synchronizes the information to other CNs, ensuring consistent job information between different CNs. In the case of CN failures, job information synchronization is reattempted by the hosting CNs, which increases job execution time. Although job information fails to be synchronized between CNs, job information can still be properly updated in the **pg_job** table on the hosting CNs, and jobs can be executed successfully. After a CN recovers, job information such as job execution time and status in its **pg_job** table may be incorrect and will be updated only after the jobs are executed again on related CNs.
5. For each job, a thread is established to execute it. If multiple jobs are triggered concurrently as scheduled, the system will need some time to start the required threads, resulting in a latency of 0.1 ms in job execution.

17.11.6 DBMS_SQL

Related Interfaces

[Table 17-38](#) lists interfaces supported by the **DBMS_SQL** package.

Table 17-38 DBMS_SQL

API	Description
DBMS_SQL.OPEN_CURSOR	Opens a cursor.
DBMS_SQL.CLOSE_CURSOR	Closes an open cursor.
DBMS_SQL.PARSE	Transmits a group of SQL statements to a cursor. Currently, only the SELECT statement is supported.
DBMS_SQL.EXECUTE	Performs a set of dynamically defined operations on the cursor.
DBMS_SQL.FETCH_ROWS	Reads a row of cursor data.
DBMS_SQL.DEFINE_COLUMN	Dynamically defines a column.
DBMS_SQL.DEFINE_COLUMN_CHAR	Dynamically defines a column of the CHAR type.
DBMS_SQL.DEFINE_COLUMN_INT	Dynamically defines a column of the INT type.
DBMS_SQL.DEFINE_COLUMN_LONG	Dynamically defines a column of the LONG type.

API	Description
DBMS_SQL.DEFINE_COLUMN_RAW	Dynamically defines a column of the RAW type.
DBMS_SQL.DEFINE_COLUMN_TEXT	Dynamically defines a column of the TEXT type.
DBMS_SQL.DEFINE_COLUMN_UNKNOWN	Dynamically defines a column of an unknown type.
DBMS_SQL.COLUMN_VALUE	Reads a dynamically defined column value.
DBMS_SQL.COLUMN_VALUE_CHAR	Reads a dynamically defined column value of the CHAR type.
DBMS_SQL.COLUMN_VALUE_INT	Reads a dynamically defined column value of the INT type.
DBMS_SQL.COLUMN_VALUE_LONG	Reads a dynamically defined column value of the LONG type.
DBMS_SQL.COLUMN_VALUE_RAW	Reads a dynamically defined column value of the RAW type.
DBMS_SQL.COLUMN_VALUE_TEXT	Reads a dynamically defined column value of the TEXT type.
DBMS_SQL.COLUMN_VALUE_UNKNOWN	Reads a dynamically defined column value of an unknown type.
DBMS_SQL.IS_OPEN	Checks whether a cursor is opened.

 NOTE

- You are advised to use **dbms_sql.define_column** and **dbms_sql.column_value** to define columns.
- If the size of the result set is greater than the value of **work_mem**, the result set will be flushed to disk. The value of **work_mem** must be no greater than 512 MB.
- **DBMS_SQL.OPEN_CURSOR**

This function opens a cursor and is the prerequisite for the subsequent **dbms_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 function prototype of **DBMS_SQL.OPEN_CURSOR** is:

```
DBMS_SQL.OPEN_CURSOR (  
    )  
RETURN INTEGER;
```
- **DBMS_SQL.CLOSE_CURSOR**

This function closes a cursor. It is the end of each **dbms_sql** operation. If this function is not invoked 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 interface in the exception handling of the stored procedure.

The function prototype of **DBMS_SQL.CLOSE_CURSOR** is:

```
DBMS_SQL.CLOSE_CURSOR (  
  cursorid IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-39 DBMS_SQL.CLOSE_CURSOR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be closed

- **DBMS_SQL.PARSE**

This function parses the query statement of a given cursor. The input query statement is executed immediately. Currently, only the **SELECT** query statement can be parsed. The statement parameters can be transferred only through the TEXT type. The length cannot exceed 1 GB.

The function prototype of **DBMS_SQL.PARSE** is:

```
DBMS_SQL.PARSE (  
  cursorid IN INTEGER,  
  query_string IN TEXT,  
  label IN INTEGER  
)  
RETURN BOOLEAN;
```

Table 17-40 DBMS_SQL.PARSE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed
query_string	Query statements to be parsed
language_flag	Version language number. Currently, only 1 is supported.

- **DBMS_SQL.EXECUTE**

This function executes a given cursor. This function receives a cursor ID. The obtained data after is used for subsequent operations. Currently, only the **SELECT** query statement can be executed.

The function prototype of **DBMS_SQL.EXECUTE** is:

```
DBMS_SQL.EXECUTE(  
  cursorid IN INTEGER,  
)  
RETURN INTEGER;
```

Table 17-41 DBMS_SQL.EXECUTE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed

- DBMS_SQL.FETCH_ROWS

This function returns the number of data rows that meet query conditions. Each time the interface is executed, the system obtains a set of new rows until all data is read.

The function prototype of **DBMS_SQL.FETCH_ROWS** is:

```
DBMS_SQL.FETCH_ROWS(  
  cursorid IN INTEGER,  
)  
RETURN INTEGER;
```

Table 17-42 DBMS_SQL.FETCH_ROWS interface parameters

Parameter Name	Description
curosorid	ID of the cursor to be executed

- DBMS_SQL.DEFINE_COLUMN

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 the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN** is:

```
DBMS_SQL.DEFINE_COLUMN(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  column_ref IN ANYELEMENT,  
  column_size IN INTEGER default 1024  
)  
RETURN INTEGER;
```

Table 17-43 DBMS_SQL.DEFINE_COLUMN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_ref	Variable of any type. You can select an appropriate interface to dynamically define columns based on variable types.
column_size	Length of a defined column

- DBMS_SQL.DEFINE_COLUMN_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 the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_CHAR** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid   IN INTEGER,  
  position   IN INTEGER,  
  column     IN TEXT,  
  column_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-44 DBMS_SQL.DEFINE_COLUMN_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter to be defined
column_size	Length of a dynamically defined column

- **DBMS_SQL.DEFINE_COLUMN_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 the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_INT** is:

```
DBMS_SQL.DEFINE_COLUMN_INT(  
  cursorid   IN INTEGER,  
  position   IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-45 DBMS_SQL.DEFINE_COLUMN_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- **DBMS_SQL.DEFINE_COLUMN_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 the input variable determines the column type. The maximum size of a long column is 1 GB.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_LONG** is:

```
DBMS_SQL.DEFINE_COLUMN_LONG(  
  cursorid IN INTEGER,  
  position IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-46 DBMS_SQL.DEFINE_COLUMN_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.DEFINE_COLUMN_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 the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_RAW** is:

```
DBMS_SQL.DEFINE_COLUMN_RAW(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  column IN BYTEA,  
  column_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-47 DBMS_SQL.DEFINE_COLUMN_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter of the RAW type
column_size	Column length

- DBMS_SQL.DEFINE_COLUMN_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 the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_TEXT** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  max_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 17-48 DBMS_SQL.DEFINE_COLUMN_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
max_size	Maximum length of the defined TEXT type

- DBMS_SQL.DEFINE_COLUMN_UNKNOWN

This function processes columns of unknown data types returned from a given cursor and is used only for the system to report an error and exist when the type cannot be identified.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_UNKNOWN** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid    IN INTEGER,  
  position    IN INTEGER,  
  column      IN TEXT  
)  
RETURN INTEGER;
```

Table 17-49 DBMS_SQL.DEFINE_COLUMN_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Dynamically defined parameter

- DBMS_SQL.COLUMN_VALUE

This function returns the cursor element value specified by a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE** is:

```
DBMS_SQL.COLUMN_VALUE(  
  cursorid    IN INTEGER,  
  position    IN INTEGER,  
  column_value INOUT ANYELEMENT  
)  
RETURN ANYELEMENT;
```

Table 17-50 DBMS_SQL.COLUMN_VALUE interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

Parameter Name	Description
column_value	Return value of a defined column

- DBMS_SQL.COLUMN_VALUE_CHAR

This function returns the value of the CHAR type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_CHAR** is:

```
DBMS_SQL.COLUMN_VALUE_CHAR(  
    cursorid      IN   INTEGER,  
    position      IN   INTEGER,  
    column_value  INOUT CHARACTER,  
    err_num       INOUT NUMERIC default 0,  
    actual_length  INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 17-51 DBMS_SQL.COLUMN_VALUE_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Return value
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is -1 regardless of the argument.
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_INT

This function returns the value of the INT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS. The function prototype of **DBMS_SQL.COLUMN_VALUE_INT** is:

```
DBMS_SQL.COLUMN_VALUE_INT(  
    cursorid      IN   INTEGER,  
    position      IN   INTEGER  
)  
RETURN INTEGER;
```

Table 17-52 DBMS_SQL.COLUMN_VALUE_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_LONG** is:

```
DBMS_SQL.COLUMN_VALUE_LONG(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  length        IN  INTEGER,  
  off_set       IN  INTEGER,  
  column_value  INOUT TEXT,  
  actual_length INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 17-53 DBMS_SQL.COLUMN_VALUE_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
length	Length of a return value
off_set	Start position of a return value
column_value	Return value
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_RAW

This function returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_RAW** is:

```
DBMS_SQL.COLUMN_VALUE_RAW(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  column_value  INOUT BYTEA,  
  err_num       INOUT NUMERIC default 0,  
  actual_length INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 17-54 DBMS_SQL.COLUMN_VALUE_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Returned column value

Parameter Name	Description
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is -1 regardless of the argument.
actual_length	Length of a return value. The value longer than this length will be truncated.

- **DBMS_SQL.COLUMN_VALUE_TEXT**

This function returns the value of the TEXT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_TEXT** is:

```
DBMS_SQL.COLUMN_VALUE_TEXT(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER  
)  
RETURN TEXT;
```

Table 17-55 DBMS_SQL.COLUMN_VALUE_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- **DBMS_SQL.COLUMN_VALUE_UNKNOWN**

This function returns the value of an unknown type in a specified position of a cursor. This is an error handling interface when the type is not unknown.

The function prototype of **DBMS_SQL.COLUMN_VALUE_UNKNOWN** is:

```
DBMS_SQL.COLUMN_VALUE_UNKNOWN(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  COLUMN_TYPE    IN  TEXT  
)  
RETURN TEXT;
```

Table 17-56 DBMS_SQL.COLUMN_VALUE_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_type	Returned parameter type

- **DBMS_SQL.IS_OPEN**

This function returns the status of a cursor: **open**, **parse**, **execute**, or **define**. The value is **TRUE**. If the status is unknown, an error is reported. In other cases, the value is **FALSE**.

The function prototype of **DBMS_SQL.IS_OPEN** is:

```
DBMS_SQL.IS_OPEN(  
    cursorid          IN    INTEGER  
)  
RETURN BOOLEAN;
```

Table 17-57 DBMS_SQL.IS_OPEN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be queried

Examples

```
-- Perform operations on RAW data in a stored procedure.  
create or replace procedure pro_dbms_sql_all_02(in_raw raw,v_in int,v_offset int)  
as  
    cursorid 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_dbms_sql_all_tb1_02 ;  
    create table pro_dbms_sql_all_tb1_02(a int ,b blob);  
    insert into pro_dbms_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEF'));  
    insert into pro_dbms_sql_all_tb1_02 values(2,in_raw);  
    query := 'select * from pro_dbms_sql_all_tb1_02 order by 1';  
    -- Open a cursor.  
    cursorid := dbms_sql.open_cursor();  
    -- Compile the cursor.  
    dbms_sql.parse(cursorid, query, 1);  
    -- Define a column.  
    define_column_ret:= dbms_sql.define_column(cursorid,1,v_id);  
    define_column_ret_raw:= dbms_sql.define_column_raw(cursorid,2,v_info,10);  
    -- Execute the cursor.  
    execute_ret := dbms_sql.execute(cursorid);  
    loop  
    exit when (dbms_sql.fetch_rows(cursorid) <= 0);  
    -- Obtain values.  
    dbms_sql.column_value(cursorid,1,v_id);  
    dbms_sql.column_value_raw(cursorid,2,v_info,v_in,v_offset);  
    -- Output the result.  
    dbms_output.put_line('id: || v_id || ' info: ' || v_info);  
    end loop;  
    -- Close the cursor.  
    dbms_sql.close_cursor(cursorid);  
end;  
/  
-- Invoke the stored procedure.  
call pro_dbms_sql_all_02(HEXTORAW('DEADBEEF'),0,1);  
  
-- Delete the stored procedure.  
DROP PROCEDURE pro_dbms_sql_all_02;
```

17.12 Debugging

Syntax

RAISE has the following five syntax formats:

Figure 17-34 raise_format::=

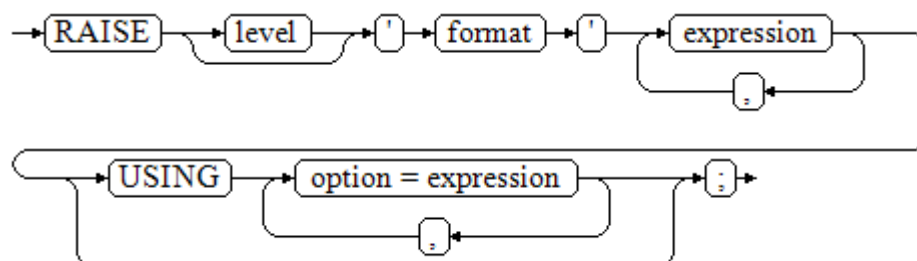


Figure 17-35 raise_condition::=

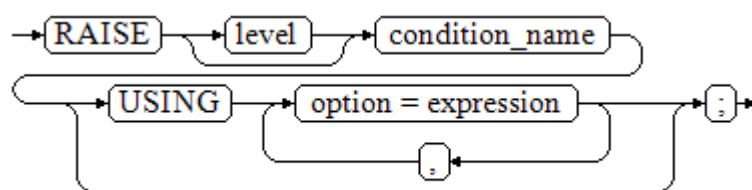


Figure 17-36 raise_sqlstate::=

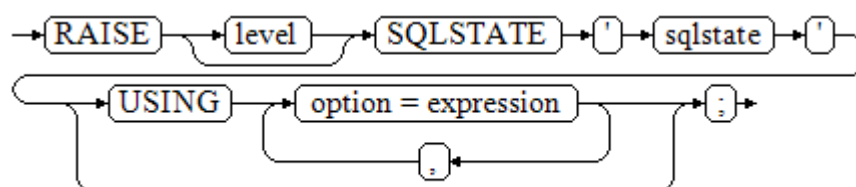


Figure 17-37 raise_option::=

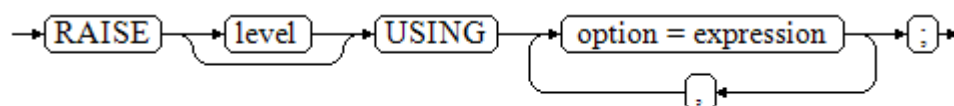
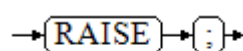


Figure 17-38 raise::=



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 [log_min_messages](#) and [client_min_messages](#) parameters control whether the error messages of specific levels are reported to the client and are written to the server log.
- **format**: specifies the error message text to be reported, a format character string. The format character string can be appended with an expression for insertion to the message text. In a format character string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:

```
--v_job_id replaces % in the character 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 character 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 designated in a **RAISE EXCEPTION** command, the **RAISE EXCEPTION (P0001)** is used by default. If no message text is designated, the condition name or SQLSTATE is used as the message text by default.

NOTICE

If the **SQLSTATE** designates 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 this kind of error codes are type codes and can be captured by the whole category.

NOTE

The syntax described in [Figure 17-38](#) 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
LOCATION: exec_stmt_raise, pl_exec.cpp:3482
```

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;
```

18 System Catalogs and System Views

18.1 Overview of System Catalogs and System Views

System catalogs store structured metadata of GaussDB(DWS). They are the source of information used by GaussDB(DWS) to control system running and are a core component of the database system.

System views provide ways to query system catalogs and internal database status.

In separation of duty, non-administrators have no permission to view system catalogs and views. In other scenarios, system catalogs and views are either visible only to administrators or visible to all users. Some of the following system catalogs and views have marked the need of administrator permissions. They are accessible only to administrators.

You can delete and recreate the 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 and system views, or rename the corresponding schemas. They are automatically maintained by the system.

NOTICE

Do not add, delete, or modify system catalogs because doing so will result in exceptions or even cluster unavailability.

18.2 System Catalogs

18.2.1 GS_OBSSCANINFO

GS_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 18-1 GS_OBSCANINFO columns

Name	Type	Reference	Description
query_id	bigint	-	Specifies a query ID.
user_id	text	-	Specifies a database user who performs queries.
table_name	text	-	Specifies the name of a foreign table on OBS.
file_type	text	-	Specifies the format of files storing the underlying data.
time_stamp	time_stamp	-	Specifies the scanning start time.
actual_time	double	-	Specifies the scanning execution time in seconds.
file_scanned	bigint	-	Specifies the number of files scanned.
data_size	double	-	Specifies the size of data scanned in bytes.
billing_info	text	-	Specifies the reserved fields.

18.2.2 GS_WLM_INSTANCE_HISTORY

The **GS_WLM_INSTANCE_HISTORY** system catalog stores information about resource usage related to CN or DN instances. Each record in the system table indicates the resource usage of an instance at a specific time point, including the memory, number of CPU cores, disk I/O, physical I/O of the process, and logical I/O of the process.

Table 18-2 GS_WLM_INSTANCE_HISTORY column

Name	Type	Description
instancename	text	Instance name
timestamp	timestamp with time zone	Timestamp
used_cpu	int	CPU usage of an instance
free_mem	int	Unused memory of an instance (unit: MB)
used_mem	int	Used memory of an instance (unit: MB)

Name	Type	Description
io_await	real	Specifies the io_await value (average value within 10 seconds) of the disk used by an instance.
io_util	real	Specifies the io_util value (average value within 10 seconds) of the disk used by an instance.
disk_read	real	Specifies the disk read rate (average value within 10 seconds) of an instance (unit: KB/s).
disk_write	real	The disk write rate (average value within 10 seconds) of an instance (unit: KB/s).
process_read	bigint	Specifies the read rate (excluding the number of bytes read from the disk pagecache) of the corresponding instance process that reads data from a disk. (Unit: KB/s)
process_write	bigint	Specifies the write rate (excluding the number of bytes written to the disk pagecache) of the corresponding instance process that writes data to a disk within 10 seconds. (Unit: KB/s)
logical_read	bigint	CN instance: N/A DN instance: Specifies the logical read byte rate of the instance in the statistical interval (10 seconds). (Unit: KB/s)
logical_write	bigint	CN instance: N/A DN instance: Specifies the logical write byte rate of the instance within the statistical interval (10 seconds). (Unit: KB/s)
read_counts	bigint	CN instance: N/A DN instance: Specifies the total number of logical read operations of the instance in the statistical interval (10 seconds).
write_counts	bigint	CN instance: N/A DN instance: Specifies the total number of logical write operations of the instance in the statistical interval (10 seconds).

18.2.3 GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO records operators of completed jobs. The data is dumped from the kernel to a system catalog.

Table 18-3 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
queryid	bigint	Internal query_id used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	plan_node_id of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	Degree of parallelism (DOP) of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the current operator
min_peak_memory	integer	Minimum peak memory used by the current operator on all DN. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DN. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DN. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DN
min_spill_size	integer	Minimum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0.
max_spill_size	integer	Maximum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0.
average_spill_size	integer	Average spilled data of all DN when a spill occurs. The unit is MB. The default value is 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. The unit is ms.

Name	Type	Description
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
warning	text	Warning. The following warnings are displayed: 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

18.2.4 GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO records load management information about a completed job executed on the current CN. The data is dumped from the kernel to a system catalog.

18.2.5 GS_WLM_USER_RESOURCE_HISTORY

The **GS_WLM_USER_RESOURCE_HISTORY** system table stores information about resources used by users and is valid only on CNs. Each record in the system table indicates the resource usage of a user at a time point, including the memory, number of CPU cores, storage space, temporary space, operator flushing space, logical I/O traffic, number of logical I/O times, and logical I/O rate. The memory, CPU, and I/O monitoring items record only the resource usage of complex jobs.

Data in the **GS_WLM_USER_RESOURCE_HISTORY** system table comes from the [PG_TOTAL_USER_RESOURCE_INFO](#) view.

Table 18-4 GS_WLM_USER_RESOURCE_HISTORY column

Name	Type	Description
username	text	Username
timestamp	timestamp with time zone	Timestamp

Name	Type	Description
used_memory	int	Specifies the used memory (unit: MB)
total_memory	int	Available memory (unit: MB). 0 indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	real	Number of CPU cores in use
total_cpu	int	Total number of CPU cores of the Cgroup associated with a user on the node
used_space	bigint	Used storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB). -1 indicates that the storage space is not limited.
used_temp_space	bigint	Used temporary storage space (unit: KB)
total_temp_space	bigint	Available temporary storage space (unit: KB). -1 indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space of operator flushing (unit: KB)
total_spill_space	bigint	Available storage space for operator flushing (unit: KB). The value -1 indicates that the maximum operator flushing space is not limited.
read_kbytes	bigint	Byte traffic of read operations in a monitoring period (unit: KB)
write_kbytes	bigint	Byte traffic of write operations in a monitoring period (unit: KB)
read_counts	bigint	Number of read operations in a monitoring period.
write_counts	bigint	Number of write operations in a monitoring period.
read_speed	real	Byte rate of read operations in a monitoring period (unit: KB)
write_speed	real	Byte rate of write operations in a monitoring period (unit: KB)

18.2.6 PGAggregate

pg_aggregate records information about aggregation 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 18-5 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	PG_PROC.oid	PG_PROC OID of the aggregate function
aggtransfn	regproc	PG_PROC.oid	Transition function
aggcollectfn	regproc	PG_PROC.oid	Aggregate function
aggfinalfn	regproc	PG_PROC.oid	Final function (zero if none)
aggstortop	oid	PG_OPERATOR.oid	Associated sort operator (zero if none)
aggtranstype	oid	PG_TYPE.oid	Data type of the aggregate function's internal transition (state) data
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 out 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 out null.

18.2.7 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 18-6 PG_AM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amname	name	-	Name of the access method

Name	Type	Reference	Description
amstrategies	smallint	-	Number of operator strategies for this access method, or zero if access method does not have a fixed set of operator strategies
amsupport	smallint	-	Number of support routines for this access method
amcanorder	Boolean	-	Whether the access method supports ordered scans sorted by the indexed column's value
amcanorderbyop	Boolean	-	Whether the access method supports ordered scans sorted by the result of an operator on the indexed column
amcanbackward	Boolean	-	Whether the access method supports backward scanning
amcanunique	Boolean	-	Whether the access method supports unique indexes
amcanmulticol	Boolean	-	Whether the access method supports multi-column indexes
amoptionalkey	Boolean	-	Whether the access method supports a scan without any constraint for the first index column
amsearcharray	Boolean	-	Whether the access method supports ScalarArrayOpExpr searches
amsearchnulls	Boolean	-	Whether the access method supports IS NULL/NOT NULL searches
amstorage	Boolean	-	Whether an index storage data type can differ from a column data type
amclusterable	Boolean	-	Whether an index of this type can be clustered on
ampredlocks	Boolean	-	Whether an index of this type manages fine-grained predicate locks
amkeytype	oid	PG_TYPE.oid	Type of data stored in index, or zero if not a fixed type
aminsert	regproc	PG_PROC.oid	"Insert this tuple" function

Name	Type	Reference	Description
ambeginscan	regproc	PG_PROC.oid	"Prepare for index scan" function
amgettupple	regproc	PG_PROC.oid	"Next valid tuple" function, or zero if none
amgetbitmap	regproc	PG_PROC.oid	"Fetch all valid tuples" function, or zero if none
amrescan	regproc	PG_PROC.oid	"(Re)start index scan" function
amendscan	regproc	PG_PROC.oid	"Clean up after index scan" function
ammarkpos	regproc	PG_PROC.oid	"Mark current scan position" function
amrestrpos	regproc	PG_PROC.oid	"Restore marked scan position" function
ammerge	regproc	PG_PROC.oid	"Merge multiple indexes" function
ambuild	regproc	PG_PROC.oid	"Build new index" function
ambuildempty	regproc	PG_PROC.oid	"Build empty index" function
ambulkdelete	regproc	PG_PROC.oid	Bulk-delete function
amvacuumclean up	regproc	PG_PROC.oid	Post- VACUUM cleanup function
amcanreturn	regproc	PG_PROC.oid	Function to check whether index supports index-only scans, or zero if none
amcostestimate	regproc	PG_PROC.oid	Function to estimate cost of an index scan
amoptions	regproc	PG_PROC.oid	Function to parse and validate reloptions for an index

18.2.8 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 18-7 PG_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amopfamily	oid	PG_OPFAMILY.oid	Operator family this entry is for
amoplefttype	oid	PG_TYPE.oid	Left-hand input data type of operator
amoprightrighttype	oid	PG_TYPE.oid	Right-hand input data type of operator
amopstrategy	smallint	-	Number of operator strategies
amoppurpose	"char"	-	Operator purpose, either s for search or o for ordering
amopopr	oid	PG_OPERATOR.oid	OID of the operator
amopmethod	oid	PG_AM.oid	Index access method the operator family is for
amopsortfamily	oid	PG_OPFAMILY.oid	The btree operator family this entry sorts according to, if an ordering operator; zero if 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 the **ORDER BY** are specified by the **amopsortfamily** column, which must reference a btree operator family for the operator's result type.

18.2.9 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 belonging to an operator family.

Table 18-8 PG_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amprocfamily	oid	PG_OPFAMILY.oid	Operator family this entry is for
amproclefttype	oid	PG_TYPE.oid	Left-hand input data type of associated operator
amprocrighttype	oid	PG_TYPE.oid	Right-hand input data type of associated operator
amprocnum	smallint	-	Support procedure number
amproc	regproc	PG_PROC.oid	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**.

18.2.10 PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING records the load mapping group information in the database.

Table 18-9 PG_APP_WORKLOADGROUP_MAPPING columns

Name	Type	Description
appname	name	Application name
workload_gpname	name	Workload group name that is mapped

18.2.11 PG_ATTRDEF

PG_ATTRDEF stores default values of columns.

Table 18-10 PG_ATTRDEF columns

Name	Type	Description
adrelid	oid	Table to which the column belongs

Name	Type	Description
adnum	smallint	Number of the column
adbin	pg_node_tree	Internal representation of the default value of the column
adsrc	text	Internal representation of the readable default value

18.2.12 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 18-11 PG_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which the column belongs
attname	name	Column name
atttypid	oid	Column type
attstattarget	integer	Controls the level of details of statistics collected for this column by ANALYZE . • A zero value indicates that no statistics should be collected. • A negative value says to use the system default statistics target. • 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.typelen of the column's type
attnum	smallint	Number of a column.
attndims	integer	Number of dimensions if the column is an array; otherwise, the value is 0.
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.

Name	Type	Description
atttypmod	integer	Type-specific data supplied at 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.typbyval of the column's type
attstorage	"char"	Copy of pg_type.typstorage of this column's type
attalign	"char"	Copy of pg_type.typalign of the column's type
attnotnull	Boolean	A not-null constraint. It is possible to change this column to enable or disable the constraint.
atthasdef	Boolean	Indicates that 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	Whether the column has been dropped and is no longer valid. A dropped column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL.
attislocal	Boolean	Whether the column is defined locally in the relation. Note that a column can be locally defined and inherited simultaneously.
attcmprmode	tinyint	Compressed modes for a specific column The compressed mode includes: • ATT_CMPR_NOCOMPRESS • ATT_CMPR_DELTA • ATT_CMPR_DICTIONARY • ATT_CMPR_PREFIX • ATT_CMPR_NUMSTR
attinhcount	integer	Number of direct ancestors 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[]	Property-level options

Name	Type	Description
attfdwoptions	text[]	Property-level external data options
attinitdefval	bytea	attinitdefval stores the default value expression. ADD COLUMN in a row-store table must use this column.

18.2.13 PG_AUTHID

PG_AUTHID records information about the 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 the 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. It is accessible only to users with system administrator rights.

Table 18-12 PG_AUTHID columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
rolname	name	Role name
rolsuper	Boolean	Whether the role is the initial system administrator with the highest permission
rolinherit	Boolean	Whether the role automatically inherits permissions of roles it is a member of
rolcreatorole	Boolean	Whether the role can create more roles
rolcreatedb	Boolean	Whether the role can create databases
rolcatupdate	Boolean	Whether the role can directly update system catalogs. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users.
rolcanlogin	Boolean	Whether a role can log in, that is, whether a role can be given as the initial session authorization identifier.
rolreplication	Boolean	Indicates that the role is a replicated one (an adaptation syntax and no actual meaning).
rolauditadmin	Boolean	Indicates that the role is an audit user.
rolsystemadmin	Boolean	Indicates that the role is an administrator.

Name	Type	Description
rolconnnlimit	integer	For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.
rolpassword	text	Password (possibly encrypted); 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
rolrespool	name	Resource pool that a user can use
roluseft	Boolean	Whether a role can perform operations on foreign tables
rolparentid	oid	OID of a group user to which the user belongs
rolkind	char	Special type of user, including private users, logical cluster administrators, and common users.
rolspacelimit	Text	The storage space of the user permanent table.
roltemp space	Text	The storage space of the user temporary table.
rolspillspace	Text	The operator disk flushing space of the user.
rolnodegroup	oid	OID of a Node Group associated with a user. The Node Group must be a logical cluster.

18.2.14 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of the role. It is accessible only to users with system administrator rights.

Table 18-13 PG_AUTH_HISTORY columns

Name	Type	Description
roloid	oid	ID of the role
passwordtime	timestamp with time zone	Time of password creation and change

Name	Type	Description
rolpassword	text	Role password that is encrypted using MD5 or SHA256, or that is not encrypted

18.2.15 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership relations between roles.

Table 18-14 PG_AUTH_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member
member	oid	ID of a role that is a member of ROLEID
grantor	oid	ID of a role that grants this membership
admin_option	Boolean	Whether a member can grant membership in ROLEID to others

18.2.16 PG_CAST

PG_CAST records conversion relationships between data types.

Table 18-15 PG_CAST columns

Name	Type	Description
castsource	oid	OID of the source data type
casttarget	oid	OID of the target data type
castfunc	oid	OID of the conversion function. If the value is 0 , no conversion function is required.
castcontext	"char"	Conversion mode between the source and target data types • e indicates that only explicit conversion can be performed (using the CAST or :: syntax). • i indicates that only implicit conversion can be performed. • a indicates that both explicit and implicit conversion can be performed between data types.

Name	Type	Description
castmethod	"char"	Conversion method • f indicates that conversion is performed using the specified function in the castfunc column. • b indicates that binary forcible conversion rather than the specified function in the castfunc column is performed between data types.

18.2.17 PG_CLASS

PG_CLASS records database objects and their relations.

Table 18-16 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
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 this table's row type (the index is 0 because the index does not have pg_type record)
reloftype	oid	OID is of composite type. 0 indicates other types.
relowner	oid	Owner of the relationship
relam	oid	Specifies the 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. If such file does not exist, the value is 0 .
reltablespace	oid	Tablespace in which this relationship is stored. If its 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 this 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.

Name	Type	Description
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 this table. The OID is 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. The OID is 0 for a table other than a TOAST table.
reldeltarelid	oid	OID of a Delta table Delta tables belong to column-store tables. They store long tail data generated during data import.
reldeltaidx	oid	OID of the index for a Delta table
relcudescrelid	oid	OID of a CU description table CU description tables (Desc tables) belong to column-store tables. They control whether storage data in the HDFS table directory is visible.
relcudescidx	oid	OID of the index for a CU description table
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 cluster. Only certain system catalogs (such as pg_database) are shared.
relpersistence	"char"	• p indicates a permanent table. • u indicates a non-log table. • t indicates a temporary table.
relkind	"char"	• r indicates an ordinary table. • i indicates an index. • S indicates a sequence. • v indicates a view. • c indicates the composite type. • t indicates a TOAST table. • f indicates a foreign table.

Name	Type	Description
relnatts	smallint	Number of user columns in the relationship (excluding system columns) pg_attribute has the same number of rows corresponding to the user columns.
relchecks	smallint	Number of constraints on a table. For details, see PG_CONSTRAINT .
relhasoids	Boolean	Its value is true if an OID is generated for each row of the relationship.
relhaspkey	Boolean	Its value is true if the table has (or once had) a primary key.
relhasrules	Boolean	Its value is true if the table has rules. See table PG_REWRITE to check whether it has rules.
relhastriggers	Boolean	Its value is true if the table has (or once had) triggers. See PG_TRIGGER .
relhas subclass	Boolean	Its value is true if the table has (or once had) any inheritance child table.
relcmprs	tinyint	Whether the compression feature is enabled for the table. Note that only batch insertion triggers compression so ordinary CRUD does not trigger compression. • 0 indicates other tables that do not support compression (primarily system tables, on which the compression attribute cannot be modified). • 1 indicates that the compression feature of the table data is NOCOMPRESS or has no specified keyword. • 2 indicates that the compression feature of the table data is COMPRESS.
relhasclusterkey	Boolean	Whether the local cluster storage is used
relrowmovement	Boolean	Whether the row migration is allowed when the partitioned table is updated • true indicates that the row migration is allowed. • false indicates that the row migration is not allowed.

Name	Type	Description
parttype	"char"	Whether the table or index has the property of a partitioned table • p indicates that the table or index has the property of a partitioned table. • n indicates that the table or index does not have the property of a partitioned table. • v indicates that the table is the value partitioned table in the HDFS.
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order 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 privileges to a role =xxxx/yyyy --Assigning the permission to public xxxx indicates the assigned privileges, and yyyy indicates the roles that are assigned to the privileges. For details about permission descriptions, see Table 18-17 .
reloptions	text[]	Index access method, using character strings in format of "keyword=value".
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order 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.

Table 18-17 Description of privileges

Parameter	Description
r	SELECT (read)
w	UPDATE (write)

Parameter	Description
a	INSERT (insert)
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
arwdDxt	ALL PRIVILEGES (used for tables)
*	Authorization options for preceding permissions

18.2.18 PG_COLLATION

PG_COLLATION records the available collations, which are essentially mappings from an SQL name to operating system locale categories.

Table 18-18 PG_COLLATION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
collname	name	-	Collation name (unique per namespace and encoding)
collnamespace	oid	PG_NAMESPACE .oid	OID of the namespace that contains this collation
collowner	oid	PG_AUTHID .oid	Owner of the collation
collencoding	integer	-	Encoding in which the collation is applicable, or -1 if it works for any encoding
collcollate	name	-	LC_COLLATE for this collation object
collctype	name	-	LC_CTYPE for this collation object

18.2.19 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, unique, and foreign key constraints on the tables.

Table 18-19 PG_CONSTRAINT columns

Name	Type	Description
conname	name	Constraint name (not necessarily unique)
connamespace	oid	OID of the namespace that contains the constraint
contype	"char"	• c indicates check constraints. • f indicates foreign key constraints. • p indicates primary key constraints. • u indicates unique constraints. • t indicates trigger constraints.
condeferrable	Boolean	Whether the constraint can be deferrable
condeferred	Boolean	Whether the constraint can be deferrable by default
convalidated	Boolean	Whether the constraint is valid Currently, only foreign key and check constraints can be set to false.
conrelid	oid	Table containing this constraint. The value is 0 if it is not a table constraint.
contypid	oid	Domain containing this constraint. The value is 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 .
confupdtype	"char"	Foreign key update action code • a indicates no action. • r indicates restriction. • c indicates cascading. • n indicates that the parameter is set to null. • d indicates that the default value is used.

Name	Type	Description
confdeltype	"char"	Foreign key deletion action code • a indicates no action. • r indicates restriction. • c indicates cascading. • n indicates that the parameter is set to null. • d indicates that the default value is used.
confmatchtype	"char"	Foreign key match type • f indicates full match. • p indicates partial match. • u indicates simple match (not specified).
conislocal	Boolean	Whether the local constraint is defined for the relationship
coninhcount	integer	Number of direct inheritance parent tables this constraint has. When the number is not 0, the constraint cannot be deleted or renamed.
connoinherit	Boolean	Whether the constraint can be inherited
consoft	Boolean	Whether the column indicates an informational constraint.
conopt	Boolean	Whether you can use Informational Constraint to optimize the execution plan.
conkey	smallint[]	Column list of the constrained control if this column is a table constraint
confkey	smallint[]	List of referenced columns if this column is a foreign key
confpeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key
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

Name	Type	Description
consrc	text	Human-readable representation of the expression if this column is a check constraint

NOTICE

- **consrc** is not updated when referenced objects change; for example, it will not track renaming of columns. Rather than relying on this field, it's best to use **pg_get_constraintdef()** to extract the definition of a check constraint.
- **pg_class.relchecks** must be consistent with the number of check-constraint entries in this table for each relationship.

18.2.20 PG_CONVERSION

PG_CONVERSION records encoding conversion information.

Table 18-20 PG_CONVERSION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
conname	name	-	Conversion name (unique in a namespace)
connamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains this conversion
conowner	oid	PG_AUTHID.oid	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID
conproc	regproc	PG_PROC.oid	Conversion procedure
condefault	Boolean	-	Its value is true if this is the default conversion.

18.2.21 PG_DATABASE

PG_DATABASE records information about the available databases.

Table 18-21 PG_DATABASE columns

Name	Type	Description
datname	name	Database name
datdba	oid	Owner of the database, usually the user who created it
encoding	integer	Character encoding for this database
datcollate	name	Sequence used by the database
datctype	name	Character type used by the database
datistemplate	Boolean	Whether this column can serve as a template database
datallowconn	Boolean	If false then no one can connect to this database. This column is used to protect the template0 database from being altered.
datconnlimit	integer	Maximum number of concurrent connections allowed on this database. -1 indicates no limit.
datlastsysoid	oid	Last system OID in the database
datfrozenxid	xid32	Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound. 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
datacl	aclitem[]	Access permissions
datfrozenxid64	xid	Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound.

18.2.22 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING records the default values of configuration items bonded to each role and database when the database is running.

Table 18-22 PG_DB_ROLE_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items; the value is 0 if the database is not specified
setrole	oid	Role corresponding to the configuration items; the value is 0 if the role is not specified
setconfig	text[]	Default value of configuration items when the database is running

18.2.23 PG_DEFAULT_ACL

PG_DEFAULT_ACL records the initial privileges assigned to the newly created objects.

Table 18-23 PG_DEFAULT_ACL columns

Name	Type	Description
defaclrole	oid	ID of the role associated with the permission
defaclnamespace	oid	Namespace associated with the permission; the value is 0 if no ID
defaclobjtype	"char"	Object type of the permission
defaclacl	aclitem[]	Access permissions that this type of object should have on creation

18.2.24 PG_DEPEND

PG_DEPEND records the dependency relationships 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 provides a similar function for dependencies involving objects that are shared across a database cluster.

Table 18-24 PG_DEPEND columns

Name	Type	Reference	Description
classid	oid	PG_CLASS .oid	OID of the system catalog the dependent object is in
objid	oid	Any OID column	OID of the specific dependent object

Name	Type	Reference	Description
objsubid	integer	-	For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is 0 .
refclassid	oid	PG_CLASS .oid	OID of the system catalog the referenced object is in
refobjid	oid	Any OID column	OID of the specific referenced object
refobjsubid	integer	-	For a table column, this is the column number (the refobjid and refclassid refer to the table itself). For all other object types, this column is 0 .
deptype	"char"	-	A code defining the specific semantics of this dependency relationship

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 defined 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): dependent objects depended object extension of a member. For details, see [PG_EXTENSION](#)). The dependent object can be dropped 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.

18.2.25 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 performs a similar function for descriptions involving objects that are shared across a database cluster.

Table 18-25 PG_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this description pertains to
classoid	oid	PG_CLASS oid	OID of the system catalog this object appears in
objsubid	integer	-	For a comment on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is 0 .
description	text	-	Arbitrary text that serves as the description of this object

18.2.26 PG_DIRECTORY

PG_DIRECTORY records directories added by users. You can run the **CREATE DIRECTORY** statement to add directories to the system catalog. Currently, only system administrators can perform this operation.

Table 18-26 PG_DIRECTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
dirname	name	Name of a directory object
owner	oid	Owner of a directory object
dirpath	text	Directory path
diracl	aclitem[]	Access permissions

18.2.27 PG_ENUM

PG_ENUM records entries showing the values and labels for each enum type. The internal representation of a given enum value is actually the OID of its associated row in **pg_enum**.

Table 18-27 PG_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
enumtypid	oid	PG_TYPE.oid	OID of the pg_type entry that contains this enum value
enumsortorder	real	-	Sort position of this enum value within its enum type
enumlabel	name	-	Textual label for this enum 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 enum type. That is, if two even OIDs belong to the same enum 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 enum comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enum types attempt to assign even OIDs to enum values whenever possible.

When an enum type is created, its members are assigned sort-order positions from 1 to *n*. But 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 enum type.

18.2.28 PG_EXTENSION

PG_EXTENSION records information about the installed extensions. By default, GaussDB(DWS) has 12 extensions, that is, PLPGSQL, DIST_FDW, FILE_FDW, HDFS_FDW, HSTORE, PLDBGAPI, DIMSEARCH, PACKAGES, GC_FDW, UUID-OSSP, LOG_FDW, and ROACH_API.

Table 18-28 PG_EXTENSION

Name	Type	Description
extname	name	Extension name
extowner	oid	Owner of the extension
extnamespace	oid	Namespace containing the extension's exported objects

Name	Type	Description
extrelocatable	Boolean	Its value is true if the extension can be relocated to another schema.
extversion	text	Version number of the extension
extconfig	oid[]	Configuration information about the extension
extcondition	text[]	Filter conditions for the extension's configuration information

18.2.29 PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE records information about external data source. An external data source contains information about an external database, such as its password encoding. It is mainly used with Extension Connector.

Table 18-29 PG_EXTENSION_DATA_SOURCE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
srcname	name	-	Name of an external data source
srcowner	oid	PG_AUTH ID.oid	Owner of an external data source
srctype	text	-	Type of an external data source. It is NULL by default.
srcversion	text	-	Type of an external data source. It is NULL by default.
srcacl	aclitem[]	-	Access permissions
srcoptions	text[]	-	Option used for foreign data sources. It is a keyword=value string.

18.2.30 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER records foreign-data wrapper definitions. A foreign-data wrapper is the mechanism by which external data, residing on foreign servers, is accessed.

Table 18-30 PG_FOREIGN_DATA_WRAPPER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
fdwname	name	-	Name of the foreign-data wrapper
fdwowner	oid	PG_AUTHID.oid	Owner of the foreign-data wrapper
fdwhandler	oid	PG_PROC.oid	References a handler function that is responsible for supplying execution routines for the foreign-data wrapper. Its value is 0 if no handler is provided.
fdwvalidator	oid	PG_PROC.oid	References a validator function that is responsible for checking the validity of the options given to the foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. Its value is 0 if no validator is provided.
fdwacl	aclitem[]	-	Access permissions
fdwoptions	text[]	-	Option used for foreign data wrappers. It is a keyword=value string.

18.2.31 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER records the 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 18-31 PG_FOREIGN_SERVER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
srvname	name	-	Name of the foreign server
srvowner	oid	PG_AUTHID.oid	Owner of the foreign server
srvfdw	oid	PG_FOREIGN_DATA_WRAPPER.oid	OID of the foreign-data wrapper of this foreign server

Name	Type	Reference	Description
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. It is a keyword=value string.

18.2.32 PG_FOREIGN_TABLE

PG_FOREIGN_TABLE records auxiliary information about foreign tables.

Table 18-32 PG_FOREIGN_TABLE columns

Name	Type	Description
ftrelid	oid	OID of the foreign table
ftserver	oid	OID of the server where the foreign table is located
ftwriteonly	Boolean	Whether data can be written in the foreign table
ftoptions	text[]	Foreign table options

18.2.33 PG_INDEX

PG_INDEX records part of the information about indexes. The rest is mostly in **PG_CLASS**.

Table 18-33 PG_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the pg_class entry for this index
indrelid	oid	OID of the pg_class entry for the table this index is for
indnatts	smallint	Number of columns in an index
indisunique	Boolean	This index is a unique index if the value is true .

Name	Type	Description
indisprimary	Boolean	This index represents the primary key of the table if the value is true . If this value is true , the value of indisunique is true.
indisexclusion	Boolean	This index supports exclusion constraints if the value is true .
indimmediate	Boolean	A uniqueness check is performed upon data insertion if the value is true .
indisclustered	Boolean	The table was last clustered on this index if the value is true .
indisusable	Boolean	This index supports insert/select if the value is true .
indisvalid	Boolean	This index is valid for queries if the value is true . If this column is false , this index is possibly incomplete and must still be modified by INSERT/UPDATE operations, but it cannot safely be used for queries. If it is a unique index, the uniqueness property is also not true.
indcheckxmin	Boolean	If the value is true , queries must not use the index until the xmin of this row in pg_index is below their TransactionXmin event horizon, because the table may contain broken HOT chains with incompatible rows that they can see.
indisready	Boolean	If the value is true , this index is ready for inserts. If the value is false , this index is ignored when data is inserted or modified.
indkey	int2vector	This is an array of indnatts values that indicate which table columns this index creates. For example, a value of 1 3 means that the first and the third columns make up the index key. 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 column contains the OID of the operator class to use. For details, see PG_OPCLASS .
indoption	int2vector	Array of values that store per-column flag bits. The meaning of the bits is defined by the index's access method.

Name	Type	Description
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 . 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, the value is null.

18.2.34 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 18-34 PG_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	PG_CLASS.oid	OID of the child table
inhparent	oid	PG_CLASS.oid	OID of the 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.

18.2.35 PG_JOB

PG_JOB records detailed information about jobs created by users. Dedicated threads poll the pg_job table and trigger jobs based on scheduled job execution time, and update job status in the pg_job table. This table belongs to the Shared Relation category. All job records are visible to all databases.

Table 18-35 PG_JOB columns

Name	Type	Description
job_id	bigint	Job ID, primary key, unique (with a unique index)

Name	Type	Description
current_postgres_pid	bigint	If the current job has been executed, the postgres 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 job 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 node on which the job will be executed
job_status	"char"	Status of the current job. The value range is 'r', 's', 'f', 'd'. The default value is 's'. The indications are as follows: Status of job step: 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 monitor 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 (that is, when 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	Scheduled time of the next job execution, accurate to millisecond
failure_count	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval
last_start_date	timestamp without time zone	Start time of the last job execution, accurate to millisecond
last_end_date	timestamp without time zone	End time of the last job execution, accurate to millisecond

Name	Type	Description
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

18.2.36 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. In the Oracle database, this part of the job information is **`varchar(4000)`**. Storing such information in the `pg_job` table 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 18-36 PG_JOB_PROC columns

Name	Type	Description
job_oid	integer	Foreign key, which is associated with <code>job_id</code> in the <code>pg_job</code> table
what	text	Job content

18.2.37 PG_LANGUAGE

PG_LANGUAGE records programming languages. You can use them and interfaces to write functions or stored procedures.

Table 18-37 PG_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
lanname	name	-	Name of the language
lanowner	oid	PG_AUTHID .oid	Owner of the language

Name	Type	Reference	Description
lanispl	Boolean	-	The value is false for internal languages (such as SQL) and true for user-defined languages. Currently, gs_dump still uses this to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.
lanpltrusted	Boolean	-	Its value is true if this is a trusted language, which means that it is believed not to grant access to anything outside the normal SQL execution environment. Only the initial user can create functions in untrusted languages.
lanplcallfoid	oid	PG_PROC.oid	For external languages, this references the language handler, which is a special function that is responsible for executing all functions that are written in the particular language.
laninline	oid	PG_PROC.oid	This references a function that is responsible for executing "inline" anonymous code blocks (DO blocks). The value is 0 if inline blocks are not supported.
lanvalidator	oid	PG_PROC.oid	This references a language validator function that is 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 permissions

18.2.38 PG_LARGEOBJECT

PG_LARGEOBJECT records the 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 (which is currently BLCKSZ/4, or typically 2 kB).

It is accessible only to users with system administrator rights.

Table 18-38 PG_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	PG_LARGEOBJECT_METADATA.oid	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	-	Actual 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.

18.2.39 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 18-39 PG_LARGEOBJECT_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
lomowner	oid	PG_AUTHID.oid	Owner of the large object
lomacl	aclitem[]	-	Access permissions

18.2.40 PG_NAMESPACE

PG_NAMESPACE records the namespaces, that is, schema-related information.

Table 18-40 PG_NAMESPACE columns

Name	Type	Description
nspname	name	Name of the namespace
nspowner	oid	Owner of the namespace

Name	Type	Description
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 permissions For details, see GRANT and REVOKE.

18.2.41 PG_OBJECT

PG_OBJECT records the user creation, creation time, last modification time, and last analyzing time of objects of specified types (types existing in **object_type**).

Table 18-41 PG_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier.
object_type	"char"	Object type: • r indicates a table, which can be an ordinary table or a temporary table. • i indicates an index. • s indicates a sequence. • v indicates a view. • p indicates a stored procedure and function.
creator	oid	ID of the creator.
ctime	timestamp with time zone	Object creation time.
mtime	timestamp with time zone	The last modification time of an object. The modification operations include ALTER , GRANT , and REVOKE .
last_analyze_time	timestamp with time zone	Time when an object is analyzed for the last time.

NOTICE

- Only normal user operations are recorded. Operations before the object upgrade and during the **initdb** process cannot be recorded.
- **ctime** and **mtime** are the start time of the transaction.
- The time of object modification due to capacity expansion is also recorded.

18.2.42 PG_OBSSCANINFO

PG_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 18-42 PG_OBSSCANINFO columns

Name	Type	Reference	Description
query_id	bigint	-	Query ID
user_id	text	-	Database user who performs queries
table_name	text	-	Name of a foreign table on OBS
file_type	text	-	Format of files storing the underlying data
time_stamp	time_stam	-	Scanning start time
actual_time	double	-	Scanning execution time, in seconds
file_scanned	bigint	-	Number of files scanned
data_size	double	-	Size of data scanned, in bytes
billing_info	text	-	Reserved columns

18.2.43 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 lefthand input.

Table 18-43 PG_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
opcmethod	oid	PG_AM.oid	Index access method the operator class is for
opcname	name	-	Name of the operator class
opcnamespace	oid	PG_NAMESPACE.oid	Namespace to which the operator class belongs

Name	Type	Reference	Description
opowner	oid	PG_AUTHID.oid	Owner of the operator class
opfamily	oid	PG_OPFAMILY.oid	Operator family containing the operator class
opcintype	oid	PG_TYPE.oid	Data type that the operator class indexes
opcdefault	Boolean	-	Whether the operator class is the default for opcintype . If it is, its value is true .
opkeytype	oid	PG_TYPE.oid	Type of data stored in index, or zero if same as opcintype

An operator class's **opmethod** must match the **opfmeth** of its containing operator family. Also, there must be no more than one **pg_opclass** row having **opcdefault** true for any given combination of **opmethod** and **opcintype**.

18.2.44 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 18-44 PG_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
oprname	name	-	Name of the operator
oprnamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains this operator
opowner	oid	PG_AUTHID.oid	Owner of the operator
oprkind	"char"	-	• b: infix ("both") • l: prefix ("left") • r: postfix ("right")
oprcanmerge	Boolean	-	Whether the operator supports merge joins
oprcanhash	Boolean	-	Whether the operator supports hash joins
oprleft	oid	PG_TYPE.oid	Type of the left operand
oprright	oid	PG_TYPE.oid	Type of the right operand

Name	Type	Reference	Description
oprresult	oid	PG_TYPE.oid	Type of the result
oprcom	oid	PG_OPERATOR.oid	Commutator of this operator, if any
oprnegate	oid	PG_OPERATOR.oid	Negator of this operator, if any
oprcode	regproc	PG_PROC.oid	Function that implements this operator
oprrest	regproc	PG_PROC.oid	Restriction selectivity estimation function for this operator
oprjoin	regproc	PG_PROC.oid	Join selectivity estimation function for this operator

18.2.45 PG_OPFAMILY

PG_OPFAMILY defines operator families.

Each operator family is a collection of operators and associated support routines that implement the 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 concept allows cross-data-type operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

Table 18-45 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
opfmethod	oid	PG_AM.oid	Index access method the operator family is for
opfname	name	-	Name of the operator family
opfnamespace	oid	PG_NAMESPACE.oid	Namespace of the operator family
opfowner	oid	PG_AUTHID.oid	Owner of the operator family

The majority of the information defining an operator family is not in **PG_OPFAMILY**, but in the associated [PG_AMOP](#), [PG_AMPROC](#), and [PG_OPCLASS](#).

18.2.46 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 **PG_PARTITION** system catalog.

Table 18-46 PG_PARTITION columns

Name	Type	Description
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions
parttype	"char"	Object type • r indicates a partitioned table. • p indicates a table partition. • x indicates an index partition. • t indicates a TOAST table.
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 field.
intervalnum	integer	Reserved field.
partstrategy	"char"	Partition policy of the partitioned table. The following policies are supported: r indicates the range partition. v indicates the 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, 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

Name	Type	Description
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	Whether the index partition is available
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
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 a keyword=value string.
relfrozenxid64	xid	Frozen transaction ID

18.2.47 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 18-47 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.
tmpldbcreate	Boolean	The value is true if the language is created by the owner of the database.
tmplhandler	text	Name of the call handler function
tmplinline	text	Name of the anonymous block handler. If no name of the block handler exists, the value is null.
tmplvalidator	text	Name of the verification function. If no verification function is available, the value is null.
tmpllibrary	text	Path of the shared library that implements languages
tmplacl	aclitem[]	Access permissions for template (not yet used)

18.2.48 PG_PROC

PG_PROC records information about functions or procedures.

Table 18-48 PG_PROC columns

Name	Type	Description
proname	name	Name of the function
pronamespace	oid	OID of the namespace that contains the function
proowner	oid	Owner of the function
prolang	oid	Implementation language or call interface of the function
procost	real	Estimated execution cost
prorows	real	Estimate number of result rows
provariadic	oid	Data type of parameter element
protransform	regproc	Simplified call method for this function
proisagg	Boolean	Whether this function is an aggregate function

Name	Type	Description
proiswindow	Boolean	Whether this function is a window function
prosecdef	Boolean	Whether this function is a security definer (such as a "setuid" function)
proleakproof	Boolean	Whether this function has side effects. If no leakproof treatment is provided for parameters, the function throws errors.
proisstrict	Boolean	The function returns null if any call parameter is null. In that case the function does not actually be called at all. Functions that are not "strict" must be prepared to process null inputs.
proretset	Boolean	The function returns a set, that is, multiple values of the specified data type.
provolatile	"char"	Whether the function's result depends only on its input parameters, or is affected by outside factors - It is i for "immutable" functions, which always deliver the same result for the same inputs. - It is s for "stable" functions, whose results (for fixed inputs) do not change within a scan. - It is v for "volatile" functions, whose results may change at any time.
pronargs	smallint	Number of parameters
pronargdefaults	smallint	Number of parameters that have default values
prorettype	oid	OID of the returned parameter type
proargtypes	oidvector	Array with the data types of the function parameters. This array includes only input parameters (including INOUT parameters) and thus represents the call signature of the function.
proallargtypes	oid[]	Array with the data types of the 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 subscripting is 1-based, whereas for historical reasons, and proargtypes is subscripted from 0.

Name	Type	Description
proargmodes	"char"[]	Array with the modes of the function parameters. • i indicates IN parameters. • o indicates OUT parameters. • b indicates INOUT parameters. If all the parameters are IN parameters, this column is null. Note that subscripts of this array 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 subscripts 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 link symbol, a file name, or any body content specified when a function or stored procedure is created, depending on how a language or calling 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 permissions For details, see GRANT and REVOKE.
prodefaultargpos	int2vect or	Locations of the function default values. Not only the last few parameters have default values.
fencedmode	Boolean	Execution mode of a function, indicating whether a function is executed in fence or not fence mode. If the execution mode is fence, the function is executed in the fork process that is reworked. The default value is fence .

Name	Type	Description
proshippable	Boolean	Whether a function can be pushed down to DNs. The default value is false . - Functions of the IMMUTABLE type can always be pushed down to the DNs. - Functions of the STABLE or VOLATILE type can be pushed down to DNs only if their attribute is SHIPPABLE.
propackage	Boolean	Indicates whether the function supports overloading, which is mainly used for the Oracle style function. The default value is false .

18.2.49 PG_RANGE

PG_RANGE records information about range types.

This is in addition to the types' entries in [PG_TYPE](#).

Table 18-49 PG_RANGE columns

Name	Type	Reference	Description
rngtypid	oid	PG_TYPE.oid	OID of the range type
rngsubtype	oid	PG_TYPE.oid	OID of the element type (subtype) of this range type
rngcollation	oid	PG_COLLATION.oid	OID of the collation used for range comparisons, or 0 if none
rngsubopc	oid	PG_OPCLASS.oid	OID of the subtype's operator class used for range comparisons
rngcanonical	regproc	PG_PROC.oid	OID of the function to convert a range value into canonical form, or 0 if none
rngsubdiff	regproc	PG_PROC.oid	OID of the function to return the difference between two element values as double precision , or 0 if none

rngsubopc (plus **rngcollation**, if the element type is collatable) determines the sort ordering used by the range type. **rngcanonical** is used when the element type is discrete.

18.2.50 PG_REDACTION_COLUMN

PG_REDACTION_COLUMN records the information about the redacted columns.

Table 18-50 PG_REDACTION_COLUMN columns

Name	Type	Description
object_oid	oid	OID of the object to be redacted.
column_attrno	smallint	attrno of the redacted column.
function_type	integer	Redaction type.
function_parameters	text	Parameters when the redaction type is partial .
regexp_pattern	text	Format string when the redaction type is regexp .
regexp_replace_string	text	Replacement string when the redaction type is regexp .
regexp_position	integer	Specifies the start replacement position when the redaction type is regexp .
regexp_occurrence	integer	The number of replacements when the redaction type is regexp .
regexp_match_parameter	text	Regular control parameter when the redaction type is regexp .
column_description	text	Description of the redacted column.

18.2.51 PG_REDACTION_POLICY

PG_REDACTION_POLICY records information about the objects to be redacted.

Table 18-51 PG_REDACTION_POLICY columns

Name	Type	Description
object_oid	oid	OID of the object to be redacted.

Name	Type	Description
policy_name	name	Name of the redact policy.
enable	boolean	Policy status (enabled or disabled). NOTE The values of the enable function are as follows: • true: enabled. • false: disabled.
expression	pg_node_tree	Policy effective expression (for users).
policy_description	text	Description of a policy.

18.2.52 PG_RESOURCE_POOL

PG_RESOURCE_POOL records the information about database resource pool.

Table 18-52 PG_RESOURCE_POOL columns

Name	Type	Description
respool_name	name	Name of the resource pool
mem_percent	integer	Percentage of the memory configuration
cpu_affinity	bigint	Value of cores bound to the CPU
control_group	name	Name of the Cgroup where the resource pool is located
active_statements	integer	Maximum number of concurrent statements in the resource pool
max_dop	integer	Maximum concurrency. This is a reserved parameter.
memory_limit	name	Maximum memory of resource pool
parentid	oid	OID of the parent resource pool
io_limits	integer	Upper limit of IOPS. It is counted by ones for column storage and by 10 thousands for row storage.
io_priority	text	I/O priority set for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.

Name	Type	Description
is_foreign	boolean	Indicates whether the resource pool can be used for users outside the logical cluster. If it is set to true , the resource pool controls the resources of common users that do not belong to the current resource pool.

18.2.53 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 18-53 PG_REWRITE columns

Name	Type	Description
rulename	name	Rule name
ev_class	oid	Name of the table that uses the rule
ev_attr	smallint	Column this rule is for (always 0 to indicate the entire table)
ev_type	"char"	Event type for this rule: • 1 = SELECT • 2 = UPDATE • 3 = INSERT • 4 = DELETE
ev_enabled	"char"	Controls in which mode the rule fires • O: The rule fires in "origin" and "local" modes. • D: The rule is disabled. • R: The rule fires in "replica" mode. • A: The rule always fires.
is_instead	Boolean	Its value is true if the rule is an INSTEAD rule.
ev_qual	pg_node_tree	Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition
ev_action	pg_node_tree	Query tree (in the form of a nodeToString() representation) for the rule's action

18.2.54 PG_SECLABEL

PG_SECLABEL records security labels on database objects.

See also **PG_SHSECLABEL**, which performs a similar function for security labels of database objects that are shared across a database cluster.

Table 18-54 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	PG_CLASS .oid	OID of the system catalog that contains the object
objsubid	integer	-	For a security label on a table column, this is the column number.
provider	text	-	Label provider associated with this label
label	text	-	Security label applied to this object

18.2.55 PG_SHDEPEND

PG_SHDEPEND records the dependency relationships between database objects and shared objects, such as roles. This information allows GaussDB(DWS) to ensure that those objects are unreferenced before attempting to delete them.

See also **PG_DEPEND**, which performs a similar function for dependencies involving objects within a single database.

Unlike most system catalogs, **PG_SHDEPEND** is shared across all databases of a cluster: there is only one copy of **PG_SHDEPEND** per cluster, not one per database.

Table 18-55 PG_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	PG_DATABASE .oid	OID of the database the dependent object is in. The value is 0 for a shared object.
classid	oid	PG_CLASS .oid	OID of the system catalog the dependent object is in.
objid	oid	Any OID column	OID of the specific dependent object

Name	Type	Reference	Description
objsubid	integer	-	For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is 0 .
refclassid	oid	PG_CLASS .oid	OID of the system catalog the referenced object is in (must be a shared catalog)
refobjid	oid	Any OID column	OID of the specific referenced object
deptype	"char"	-	Code segment defining the specific semantics of this dependency relationship. See the following text for details.
objfile	text	-	Path of the user-defined C function library file.

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 defined 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 ACL (access control list, i.e., privileges list) of the dependent object. (A **SHARED_DEPENDENCY_ACL** entry is not made for the owner of the object, since the owner will have a **SHARED_DEPENDENCY_OWNER** entry anyway.)
- **SHARED_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.

18.2.56 PG_SHDESCRIPTION

PG_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with **psql's \d** commands.

See also **PG_DESCRIPTION**, which performs a similar function for descriptions involving objects within a single database.

Unlike most system catalogs, **PG_SHDESCRIPTION** is shared across all databases of a cluster. There is only one copy of **PG_SHDESCRIPTION** per cluster, not one per database.

Table 18-56 PG_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this description pertains to
classoid	oid	PG_CLASS .oid	OID of the system catalog where the object resides
description	text	-	Arbitrary text that serves as the description of this object

18.2.57 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 performs a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG_SHSECLABEL** is shared across all databases of a cluster. There is only one copy of **PG_SHSECLABEL** per cluster, not one per database.

Table 18-57 PG_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	PG_CLASS .oid	OID of the system catalog where the object resides
provider	text	-	Label provider associated with this label
label	text	-	Security label applied to this object

18.2.58 PG_STATISTIC

PG_STATISTIC records statistics about tables and index columns in a database. It is accessible only to users with system administrator rights.

Table 18-58 PG_STATISTIC columns

Name	Type	Description
starelid	oid	Table or index which 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	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, not-null data values in the column for all DNs • A value greater than zero is the actual number of distinct values. • A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.) • 0 indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in Slot N of the pg_statistic row. Value range: 1 to 5
staopN	oid	Operator used to generate the statistics stored in Slot N. For example, a histogram slot shows the < operator that defines the sort order of the data. Value range: 1 to 5
stanumbersN	real[]	Numerical statistics of the appropriate type for Slot N. The value is null if the slot kind does not involve numerical values. Value range: 1 to 5
stavaluesN	anyarray	Column data values of the appropriate type for Slot N. 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. Value range: 1 to 5

Name	Type	Description
stadndistinct	real	Number of unique non-null data values in the dn1 column - A value greater than zero is the actual number of distinct values. - A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.) 0 indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics (reserved)

18.2.59 PG_STATISTIC_EXT

PG_STATISTIC_EXT records the extended statistics of tables in a database, such as statistics of multiple columns. Statistics of expressions will be supported later. You can specify the extended statistics to be collected. It is accessible only to users with system administrator rights.

Table 18-59 PG_STATISTIC_EXT columns

Parameter	Type	Description
starelid	oid	Table or index which the described column belongs to
starelkind	"char"	Type of an object
stainherit	Boolean	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, not-null data values in the column for all DNs - A value greater than zero is the actual number of distinct values. - A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.) 0 indicates that the number of distinct values is unknown.

Parameter	Type	Description
stadndistinct	real	Number of unique non-null data values in the dn1 column - A value greater than zero is the actual number of distinct values. - A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.) 0 indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in Slot N of the pg_statistic row. Value range: 1 to 5
staopN	oid	Operator used to generate the statistics stored in Slot N. For example, a histogram slot shows the < operator that defines the sort order of the data. Value range: 1 to 5
stakey	int2vector	Array of a column ID
stanumbersN	real[]	Numerical statistics of the appropriate type for Slot N. The value is null if the slot kind does not involve numerical values. Value range: 1 to 5
stavaluesN	anyarray	Column data values of the appropriate type for Slot N. 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. Value range: 1 to 5
staexprs	pg_node_tree	Expression corresponding to the extended statistics information.

18.2.60 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 18-60 PG_SYNONYM columns

Name	Type	Description
synname	name	Synonym name.

Name	Type	Description
synnamespace	oid	OID of the namespace where the synonym is located.
synowner	oid	Owner of a 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.

18.2.61 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 18-61 PG_TABLESPACE columns

Name	Type	Description
spcname	name	Name of the tablespace
spcowner	oid	Owner of the tablespace, usually the user who created it
spcacl	aclitem[]	Access permissions For details, see GRANT and REVOKE.
spcoptions	text[]	Specifies options of the tablespace.
spcmaxsize	text	Maximum size of the available disk space, in bytes

18.2.62 PG_TRIGGER

PG_TRIGGER records the trigger information.

Name	Type	Description
tgrelid	oid	OID of the table where the trigger is located.
tgname	name	Trigger name.
tgfoid	oid	Trigger OID.
tgtype	smallint	Trigger type

Name	Type	Description
tgenabled	"char"	O : The trigger fires in "origin" or "local" mode. D : The trigger is disabled. R : The trigger fires in "replica" mode. A : The trigger always fires.
tgisinternal	boolean	Internal trigger ID. If the value is true, it indicates an internal trigger.
tgconstrrelid	oid	The table referenced by the integrity constraint
tgconstrindid	oid	Index of the integrity constraint
tgconstraint	oid	OID of the constraint trigger in the pg_constraint
tgdeferrable	boolean	The constraint trigger is of the DEFERRABLE type.
tginitdeferred	boolean	whether the trigger is of the INITIALLY DEFERRED type
tgargs	smallint	Input parameters number of the trigger function
tgattr	int2vector	Column ID specified by the trigger. If no column is specified, an empty array is used.
tgargs	bytea	Parameter transferred to the trigger
tgqual	pg_node_tree	Indicates the WHEN condition of the trigger. If the WHEN condition does not exist, the value is null.

18.2.63 PG_TS_CONFIG

PG_TS_CONFIG records 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 18-62 PG_TS_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)

Name	Type	Reference	Description
cfgname	name	-	Text search configuration name
cfgnamespace	oid	PG_NAMESPACE.oid	OID of the namespace where the configuration resides
cfgowner	oid	PG_AUTHID.oid	Owner of the configuration
cfgparser	oid	PG_TS_PARSER.oid	OID of the text search parser for this configuration
cfgoptions	text[]	-	Configuration options

18.2.64 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP records 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 18-63 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	-	A token type emitted by the configuration's parser
mapseqno	integer	-	Order in which to consult this entry
mapdict	oid	PG_TS_DICT.oid	OID of the text search dictionary to consult

18.2.65 PG_TS_DICT

PG_TS_DICT records 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 18-64 PG_TS_DICT columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
dictname	name	-	Text search dictionary name
dictnamespace	oid	PG_NAMESPACE .oid	OID of the namespace that contains the dictionary
dictowner	oid	PG_AUTHID .oid	Owner of the dictionary
dicttemplate	oid	PG_TS_TEMPLATE .oid	OID of the text search template for this dictionary
dictinitoption	text	-	Initialization option string for the template

18.2.66 PG_TS_PARSER

PG_TS_PARSER records entries defining text search parsers. A parser splits input text into lexemes and assigns a token type to each lexeme. Since a parser must be implemented by C functions, parsers can be created only by database administrators.

Table 18-65 PG_TS_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
prsname	name	-	Text search parser name
pramespace	oid	PG_NAMESPACE .oid	OID of the namespace that contains the parser
prsstart	regproc	PG_PROC .oid	OID of the parser's startup function
prstoken	regproc	PG_PROC .oid	OID of the parser's next-token function
prsend	regproc	PG_PROC .oid	OID of the parser's shutdown function
prsheadline	regproc	PG_PROC .oid	OID of the parser's headline function

Name	Type	Reference	Description
prsllextype	regproc	PG_PROC.oid	OID of the parser's lextype function

18.2.67 PG_TS_TEMPLATE

PG_TS_TEMPLATE records entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C functions, templates can be created only by database administrators.

Table 18-66 PG_TS_TEMPLATE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
tmplname	name	-	Text search template name
tmplnamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains the template
tmplinit	regproc	PG_PROC.oid	OID of the template's initialization function
tmpllexize	regproc	PG_PROC.oid	OID of the template's lexize function

18.2.68 PG_TYPE

PG_TYPE records the information about data types.

Table 18-67 PG_TYPE columns

Name	Type	Description
typname	name	Data type name
typnamespace	oid	OID of the namespace that contains this type
typowner	oid	Owner of this type

Name	Type	Description
typelen	smallint	Number of bytes in the internal representation of the type for a fixed-size type. But for a variable-length type, typelen is negative. • -1 indicates a "varlena" type (one that has a length word). • -2 indicates a null-terminated C string.
typbyval	Boolean	Whether the value of this type is passed by parameter or reference of this column. TYPBYVAL is false if the type of TYPLEN is not 1, 2, 4, or 8, because values of this type are always passed by reference of this column. TYPBYVAL can be false even the TYPLEN is passed by parameter of this column.
typtype	char	• b indicates a basic type. • c indicates a composite type (for example, a table's row type). • d indicates a domain type. • p indicates a pseudo type. For details, see typrelid and typbasetype .
typcategory	char	typcategory is an arbitrary classification of data types that is used by the parser to determine which implicit casts should be "preferred".
typispreferred	Boolean	Whether data is converted. It is true if conversion is performed when data meets the conversion rules specified by TYPCATEGORY .
typisdefined	Boolean	The value is true if the type is defined. The value is false if this is a placeholder entry for a not-yet-defined type. When it is false, nothing except the type name, namespace, and OID can be relied on.
typdelim	"char"	Character that separates two values of this type when parsing array input. Note that the delimiter is associated with the array element data type, not the array data type.
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. The value is 0 for non-composite types.

Name	Type	Description
typelem	oid	If typelem is not 0 then it identifies another row in pg_type . The current type can be subscripted like an array yielding values of type typelem . The current type can then be subscripted like an array yielding values of type typelem . A "true" array type is variable length (typlen = -1), but some fixed-length (typlen > 0) types also have nonzero typelem , for example name and point . If a fixed-length type has a typelem , its internal representation must be some 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	Indicates that the corresponding type record is available in pg_type if the value is not 0.
typinput	regproc	Input conversion function (text format)
typoutput	regproc	Output conversion function (text format)
typreceive	regproc	Input conversion function (binary format). If no input conversion function, the value is 0.
typsend	regproc	output conversion function (binary format). If no output conversion function, the value is 0.
typmodin	regproc	Type modifier input function. The value is 0 if the type does not support modifiers.
typmodout	regproc	Type modifier output function. The value is 0 if the type does not support modifiers.
typanalyze	regproc	Custom ANALYZE function. The value is 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 disk as well as most representations of the value inside PostgreSQL. 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 needed • s: short alignment (2 bytes on most machines) • i: int 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 tables, 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 indicates that values are always stored plain. • e: Value 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	Represents a NOTNULL constraint on a type. 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). The value is -1 if this type is not a domain.

Name	Type	Description
typndims	integer	Number of array dimensions for a domain that is an array (that is, typbasetype is an array type; the domain's typelem matches the base type's typelem). The value is 0 for types other than domains over array types.
typcollation	oid	Sequence rule for specified types. Sequencing is not supported if the value is 0.
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 null, typdefault must contain a human-readable version of the 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 permissions

18.2.69 PG_USER_MAPPING

PG_USER_MAPPING records the mappings from local users to remote.

It is accessible only to users with system administrator rights. You can use view [PG_USER_MAPPINGS](#) to query common users.

Table 18-68 PG_USER_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
umuser	oid	PG_AUTHID.oid	OID of the local role being mapped, 0 if the user mapping is public
umserver	oid	PG_FOREIGN_SERVER.oid	OID of the foreign server that contains this mapping
umoptions	text[]	-	Option used for user mapping. It is a keyword=value string.

18.2.70 PG_USER_STATUS

PG_USER_STATUS records the states of users that access to the database. It is accessible only to users with system administrator rights.

Table 18-69 PG_USER_STATUS columns

Name	Type	Description
roloid	oid	ID of the role
failcount	integer	Specifies the number of failed attempts.
locktime	timestamp with time zone	Time at which the role is locked
rolstatus	smallint	Role state • 0: normal • 1 indicates that the role is locked for some time because the failed login attempts exceed the threshold • 2 indicates that 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.

18.2.71 PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP records the workload group information in the database.

Table 18-70 PG_WORKLOAD_GROUP columns

Name	Type	Description
workload_gpname	name	Workload group name
respool_oid	oid	OID bound to the resource pool
act_statements	integer	Maximum number of active statements in the workload group

18.2.72 PGXC_CLASS

PGXC_CLASS records the replicated or distributed information for each table.

Table 18-71 PGXC_CLASS columns

Name	Type	Description
pcrelid	oid	Table OID
pclocatortype	"char"	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	Name of the node group
redistributed	"char"	The table has been redistributed.
redis_order	integer	Redistribution sequence
pcttnum	int2vector	Column number used as a distribution key
nodeoids	oidvector_extend	List of distributed table node OIDs
options	text	Extension status information. This is a reserved column in the system.

18.2.73 PGXC_GROUP

PGXC_GROUP records information about node groups.

Table 18-72 PGXC_GROUP columns

Name	Type	Description
group_name	name	Name of the node group
in_redistribution	"char"	Whether redistribution is required • n indicates that the Node Group is not redistributed. • y indicates the source Node Group in redistribution. • t indicates the destination Node Group in redistribution.
group_members	oidvector_extend	Node OID list of the node group
group_buckets	text	Distributed data bucket group

Name	Type	Description
is_installation	Boolean	Whether to install a sub-cluster
group_acl	aclitem[]	Access permissions
group_kind	"char"	Node Group type • i indicates an installation Node Group. • n indicates a Node Group in a common, non-logical cluster. • v indicates a Node Group in a logical cluster. • e indicates an elastic cluster.

18.2.74 PGXC_NODE

PGXC_NODE records information about cluster nodes.

Table 18-73 PGXC_NODE columns

Name	Type	Description
node_name	name	Node name
node_type	"char"	Node type C: CN D: DN
node_port	integer	Port ID of the node
node_host	name	Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.)
node_port1	integer	Port number of a replication node
node_host1	name	Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.)
hostis_primary	Boolean	Whether a switchover occurs between the primary and the standby server on the current node
nodeis_primary	Boolean	Whether the current node is preferred to execute non-query operations in the replication table
nodeis_preferred	Boolean	Whether the current node is preferred to execute queries in the replication table

Name	Type	Description
node_id	integer	Node identifier
sctp_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the data channel.
control_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the control channel.
sctp_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the data channel.
control_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the control channel.
nodeis_central	Boolean	Indicates that the current node is the central node.

18.3 System Views

18.3.1 ALL_ALL_TABLES

ALL_ALL_TABLES displays the tables or views accessible to the current user.

Table 18-74 ALL_ALL_TABLES columns

Name	Type	Description
owner	name	Owner of the table or the view
table_name	name	Name of the table or the view
tablespace_name	name	Tablespace where the table or view is located

18.3.2 ALL_CONSTRAINTS

ALL_CONSTRAINTS displays information about constraints accessible to the current user.

Table 18-75 ALL_CONSTRAINTS columns

Name	Type	Description
constraint_name	vcharacter 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 constraint-related table
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

18.3.3 ALL_CONS_COLUMNS

ALL_CONS_COLUMNS displays information about constraint columns accessible to the current user.

Table 18-76 ALL_CONS_COLUMNS columns

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

18.3.4 ALL_COL_COMMENTS

ALL_COL_COMMENTS displays the comment information about table columns accessible to the current user.

Table 18-77 ALL_COL_COMMENTS columns

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table name
owner	character varying(64)	Table owner
comments	text	Comments

18.3.5 ALL_DEPENDENCIES

ALL_DEPENDENCIES displays dependencies between functions and advanced packages accessible to the current user.

NOTICE

Currently in GaussDB(DWS), this table is empty without any record due to information constraints.

Table 18-78 ALL_DEPENDENCIES columns

Name	Type	Description
owner	character varying(30)	Owner of the object
name	character varying(30)	Object name
type	character varying(17)	Type of the object
referenced_owner	character varying(30)	Owner of the referenced object
referenced_name	character varying(64)	Name of the referenced object
referenced_type	character varying(17)	Type of the referenced object
referenced_link_name	character varying(128)	Name of the link to the referenced object
schemaid	numeric	ID of the current schema
dependency_type	character varying(4)	Dependency type (REF or HARD)

18.3.6 ALL_IND_COLUMNS

ALL_IND_COLUMNS displays all index columns accessible to the current user.

Table 18-79 ALL_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 column in the index

18.3.7 ALL_IND_EXPRESSIONS

ALL_IND_EXPRESSIONS displays information about the expression indexes accessible to the current user.

Table 18-80 ALL_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
column_position	smallint	Position of a column in the index

18.3.8 ALL_INDEXES

ALL_INDEXES displays information about indexes accessible to the current user.

Table 18-81 ALL_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name

Name	Type	Description
table_name	character varying(64)	Name of the table corresponding to the index.
uniqueness	text	Whether the index is a unique index
generated	character varying(1)	Whether the index name is generated by the system
partitioned	character(3)	Whether the index has the property of the partition table

18.3.9 ALL_OBJECTS

ALL_OBJECTS displays all database objects accessible to the current user.

Table 18-82 ALL_OBJECTS columns

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object
namespace	oid	ID of the namespace where the object resides
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.10 ALL PROCEDURES

ALL PROCEDURES displays information about all stored procedures or functions accessible to the current user.

Table 18-83 ALL_PROCEDURES columns

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name

18.3.11 ALL_SEQUENCES

ALL_SEQUENCES displays all sequences accessible to the current user.

Table 18-84 ALL_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of the sequence
sequence_name	name	Name of the sequence
min_value	bigint	Minimum value of the sequence
max_value	bigint	Maximum value of the sequence
increment_by	bigint	Value by which the sequence is incremented
cycle_flag	character(1)	Whether the sequence is a cycle sequence. The value can be Y or N . • Y indicates that it is a cycle sequence. • N indicates that it is not a cycle sequence.

18.3.12 ALL_SOURCE

ALL_SOURCE displays information about stored procedures or functions accessible to the current user, and provides the columns defined by the stored procedures and functions.

Table 18-85 ALL_SOURCE columns

Name	Type	Description
owner	name	Owner of the object
name	name	Object name
type	name	Type of the object
text	text	Definition of the object

18.3.13 ALL_SYNONYMS

ALL_SYNONYMS displays all synonyms accessible to the current user.

Table 18-86 ALL_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.
table_schema_name	text	Schema name of the associated object.
table_name	text	Name of the associated object.

18.3.14 ALL_TAB_COLUMNS

ALL_TAB_COLUMNS displays description information about columns of the tables accessible to the current user.

Table 18-87 ALL_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Owner of the table
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of the column
column_id	integer	Column ID generated when the object is created or column is added
data_length	integer	Length of the column in the unit of bytes
avg_col_len	numeric	Average length of a column in the unit of bytes
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n.

Name	Type	Description
data_precision	integer	Indicates the precision of the data type. This parameter is valid for the numeric data type and NULL for other types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type. For other data types, the value of this parameter is 0.
char_length	numeric	Column length in the unit of bytes. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.

18.3.15 ALL_TAB_COMMENTS

ALL_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

Table 18-88 ALL_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of the table or the view
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

18.3.16 ALL_TABLES

ALL_TABLES displays all the tables accessible to the current user.

Table 18-89 ALL_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace that contains the table
status	character varying(8)	Whether the current record is valid

Name	Type	Description
temporary	character(1)	Whether the table is a temporary table • Y indicates that it is a temporary table. • N indicates that it is not a temporary table.
dropped	character varying	Whether the current record is deleted • YES indicates that it is deleted. • NO indicates that it is not deleted.
num_rows	numeric	The estimated number of rows in the table

18.3.17 ALL_USERS

ALL_USERS displays all users of the database visible to the current user, however, it does not describe the users.

Table 18-90 ALL_USERS columns

Name	Type	Description
username	name	User name
user_id	oid	OID of the user

18.3.18 ALL_VIEWS

ALL_VIEWS displays the description about all views accessible to the current user.

Table 18-91 ALL_VIEWS columns

Name	Type	Description
owner	name	Owner of the view
view_name	name	Name of the view
text_length	integer	Text length of the view
text	text	Text in the view

18.3.19 DBA_DATA_FILES

DBA_DATA_FILES displays the description of database files. It is accessible only to users with system administrator rights.

Table 18-92 DBA_DATA_FILES columns

Name	Type	Description
tablespace_name	name	Name of the tablespace to which the file belongs
bytes	double precision	Length of the file in bytes

18.3.20 DBA_USERS

DBA_USERS displays all user names in the database. It is accessible only to users with system administrator rights.

Table 18-93 DBA_USERS columns

Name	Type	Description
username	character varying(64)	User name

18.3.21 DBA_COL_COMMENTS

DBA_COL_COMMENTS displays information about table column comments in the database. It is accessible only to users with system administrator rights.

Table 18-94 DBA_COL_COMMENTS columns

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table name
owner	character varying(64)	Table owner
comments	text	Comments

18.3.22 DBA_CONSTRAINTS

DBA_CONSTRAINTS displays information about table constraints in database. It is accessible only to users with system administrator rights.

Table 18-95 DBA_CONSTRAINTS columns

Name	Type	Description
constraint_name	vcharacter 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 constraint-related table
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

18.3.23 DBA_CONS_COLUMNS

DBA_CONS_COLUMNS displays information about constraint columns in database tables. It is accessible only to users with system administrator rights.

Table 18-96 DBA_CONS_COLUMNS columns

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

18.3.24 DBA_IND_COLUMNS

DBA_IND_COLUMNS displays column information about all indexes in the database. It is accessible only to users with system administrator rights.

Table 18-97 DBA_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 column in the index

18.3.25 DBA_IND_EXPRESSIONS

DBA_IND_EXPRESSIONS displays the information about expression indexes in the database. It is accessible only to users with system administrator rights.

Table 18-98 DBA_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	The function-based index expression of a specified column
column_position	smallint	Position of column in the index

18.3.26 DBA_IND_PARTITIONS

DBA_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 **DBA_IND_PARTITIONS**. It is accessible only to users with system administrator rights.

Table 18-99 DBA_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned index to which the index partition belongs
schema	character varying(64)	Schema of the partitioned index to which the index partition belongs
index_name	character varying(64)	Index name of the partitioned table to which the index partition belongs
partition_name	character varying(64)	Name of the index partition
index_partition_usable	Boolean	Whether the index partition is available
high_value	text	Upper boundary of the partition corresponding to the index partition
def_tablespace_name	name	Tablespace name of the index partition

18.3.27 DBA_INDEXES

DBA_INDEXES displays all indexes in the database. It is accessible only to users with system administrator rights.

Table 18-100 DBA_INDEXES columns

Name	Type	Description
owner	character varying(64)	Owner of the index
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of the table corresponding to the index
uniqueness	text	Whether the index is a unique index
generated	character varying(1)	Whether the index name is generated by the system

Name	Type	Description
partitioned	character(3)	Whether the index has the property of the partition table

18.3.28 DBA_OBJECTS

DBA_OBJECTS displays all database objects in the database. It is accessible only to users with system administrator rights.

Table 18-101 DBA_OBJECTS columns

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object
namespace	oid	Namespace containing the object
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.29 DBA_PART_INDEXES

DBA_PART_INDEXES displays information about all partitioned table indexes in the database. It is accessible only to users with system administrator rights.

Table 18-102 DBA_PART_INDEXES columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index

Name	Type	Description
schema	character varying(64)	Schema of the partitioned table index
index_name	character varying(64)	Name of the partitioned table index
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs
partitioning_type	text	Partition policy of the partitioned table NOTE Currently, only range partitioning is supported.
partition_count	bigint	Number of index partitions of the partitioned table index
def_tablespace_name	name	Tablespace name of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table

18.3.30 DBA_PART_TABLES

DBA_PART_TABLES displays information about all partitioned tables in the database. It is accessible only to users with system administrator rights.

Table 18-103 DBA_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Name of the owner of the partitioned table
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Name of the partitioned table
partitioning_type	text	Partition policy of the partitioned table NOTE Currently, only range partitioning is supported.
partition_count	bigint	Number of partitions of the partitioned table

Name	Type	Description
def_tablespace_name	name	Tablespace name of the partitioned table
partitioning_key_count	integer	Number of partition keys of the partitioned table

18.3.31 DBA_PROCEEDURES

DBA_PROCEEDURES displays information about all stored procedures and functions in the database. It is accessible only to users with system administrator rights.

Table 18-104 DBA_PROCEEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
object_name	character varying(64)	Name of the stored procedure or the function
argument_number	smallint	Number of the input parameters in the stored procedure

18.3.32 DBA_SEQUENCES

DBA_SEQUENCES displays information about all sequences in the database. It is accessible only to users with system administrator rights.

Table 18-105 DBA_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of the sequence
sequence_name	character varying(64)	Name of the sequence

18.3.33 DBA_SOURCE

DBA_SOURCE displays all stored procedures or functions in the database, and it provides the columns defined by the stored procedures or functions. It is accessible only to users with system administrator rights.

Table 18-106 DBA_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
name	character varying(64)	Name of the stored procedure or the function
text	text	Definition of the stored procedure or the function

18.3.34 DBA_SYNONYMS

DBA_SYNONYMS displays all synonyms in the database. It is accessible only to users with system administrator rights.

Table 18-107 DBA_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.
table_schema_name	text	Schema name of the associated object.
table_name	text	Name of the associated object.

18.3.35 DBA_TAB_COLUMNS

DBA_TAB_COLUMNS displays the columns of tables. Each column of a table in the database has a row in **DBA_TAB_COLUMNS**. It is accessible only to users with system administrator rights.

Table 18-108 DBA_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
column_id	integer	Sequence number of the column when the table is created
data_length	integer	Length of the column in the unit of bytes
comments	text	Comments
avg_col_len	numeric	Average length of a column in the unit of bytes
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n.
data_precision	integer	Indicates the precision of the data type. This parameter is valid for the numeric data type, however its value is NULL for other types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type. For other data types, the value of this parameter is 0.
char_length	numeric	Column length (in the unit of bytes) which is valid only for varchar, nvarchar2, bpchar, and char types.

18.3.36 DBA_TAB_COMMENTS

DBA_TAB_COMMENTS displays comments about all tables and views in the database. It is accessible only to users with system administrator rights.

Table 18-109 DBA_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of the table or the view

Name	Type	Description
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

18.3.37 DBA_TAB_PARTITIONS

DBA_TAB_PARTITIONS displays information about all partitions in the database.

Table 18-110 DBA_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Owner of the table that contains the partition
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Table name
partition_name	character varying(64)	Name of the partition
high_value	text	Upper boundary of the range partition and interval partition
tablespace_name	name	Name of the tablespace that contains the partition

18.3.38 DBA_TABLES

DBA_TABLES displays all tables in the database. It is accessible only to users with system administrator rights.

Table 18-111 DBA_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace that contains the table
status	character varying(8)	Whether the current record is valid

Name	Type	Description
temporary	character(1)	Whether the table is a temporary table • Y indicates that it is a temporary table. • N indicates that it is not a temporary table.
dropped	character varying	Whether the current record is deleted • YES indicates that it is deleted. • NO indicates that it is not deleted.
num_rows	numeric	The estimated number of rows in the table

18.3.39 DBA_TABLESPACES

DBA_TABLESPACES displays information about available tablespaces. It is accessible only to users with system administrator rights.

Table 18-112 DBA_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Name of the tablespace

18.3.40 DBA_TRIGGERS

DBA_TRIGGERS displays information about triggers in the database. It is accessible only to users with system administrator rights.

Table 18-113 DBA_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_name	character varying(64)	Name of the table that defines the trigger
table_owner	character varying(64)	Owner of the table that defines the trigger

18.3.41 DBA_VIEWS

DBA_VIEWS displays views in the database. It is accessible only to users with system administrator rights.

Table 18-114 DBA_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of the view
view_name	character varying(64)	View name

18.3.42 DUAL

DUAL 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. It is accessible to all users.

Table 18-115 DUAL columns

Name	Type	Description
dummy	text	Expression calculation result

18.3.43 GLOBAL_WORKLOAD_SQL_COUNT

GLOBAL_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in all workload Cgroups in a cluster, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 18-116 GLOBAL_WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload Cgroup 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

Name	Type	Description
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements

18.3.44 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME

GLOBAL_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in all workload Cgroups in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 18-117 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload Cgroup name
total_select_elapse	bigint	Total response time of SELECT
max_select_elapse	bigint	Maximum response time of SELECT
min_select_elapse	bigint	Minimum response time of SELECT
avg_select_elapse	bigint	Average response time of SELECT
total_update_elapse	bigint	Total response time of UPDATE
max_update_elapse	bigint	Maximum response time of UPDATE
min_update_elapse	bigint	Minimum response time of UPDATE
avg_update_elapse	bigint	Average response time of UPDATE
total_insert_elapse	bigint	Total response time of INSERT
max_insert_elapse	bigint	Maximum response time of INSERT

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of INSERT
avg_insert_elapse	bigint	Average response time of INSERT
total_delete_elapse	bigint	Total response time of DELETE
max_delete_elapse	bigint	Maximum response time of DELETE
min_delete_elapse	bigint	Minimum response time of DELETE
avg_delete_elapse	bigint	Average response time of DELETE

18.3.45 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO displays all Cgroup information in a database.

Table 18-118 GS_ALL_CONTROL_GROUP_INFO columns

Name	Type	Description
name	text	Name of the Cgroup
type	text	Type of the Cgroup
gid	bigint	Cgroup ID
classgid	bigint	ID of the Class Cgroup to which a Workload belongs
class	text	Class Cgroup
workload	text	Workload Cgroup
shares	bigint	CPU quota allocated to a Cgroup
limits	bigint	Limit of CPUs allocated to a Cgroup
wdlevel	bigint	Workload Cgroup level
cpucore	text	Usage of CPU cores in a Cgroup

18.3.46 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO displays a DN resource summary.

Table 18-119 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 a DN
min_cpu_util	integer	Minimum CPU usage of a DN
max_cpu_util	integer	Maximum CPU usage of a DN
min_io_util	integer	Minimum I/O usage of a DN
max_io_util	integer	Maximum I/O usage of a DN
used_mem_rate	integer	Maximum physical memory usage

18.3.47 GS_INSTR_UNIQUE_SQL

Unique SQL Definition

The database parses each received SQL text string and generates an internal parsing tree. The database traverses the parsing tree and ignores constant values in the parsing tree. In this case, an integer value is calculated using a certain algorithm. This integer is used as the Unique SQL ID to uniquely identify this type of SQL. SQLs with the same Unique SQL ID are called Unique SQLs.

Examples

Assume that the user enters the following SQL statements in sequence:

```
select * from t1 where id = 1;  
select * from t1 where id = 2;
```

The statistics of the two SQL statements are aggregated to the same Unique SQL statement.

```
select * from t1 where id = ?;
```

GS_INSTR_UNIQUE_SQL View

The **GS_INSTR_UNIQUE_SQL** view displays the execution information about the Unique SQL statements collected by the current node, including:

- Unique SQL ID and normalized SQL text string. The normalized SQL text is described in [Examples](#).
- Number of execution times (number of successful execution times) and response time (SQL execution time in the database, including the maximum, minimum, and total time)
- Cache/IO information, including the number of physical reads and logical reads of a block. Only information about successfully executed SQL statements on each DN is collected. The statistical value is related to factors such as the amount of data processed during query execution, used memory, whether the query is executed for multiple times, memory management

policy, and whether there are other concurrent queries. The statistical value reflects the number of physical reads and logical reads of the buffer block in the entire query execution process. The statistical value may vary according to the execution time.

- Row activities, such as the number of returned rows, updated rows, inserted rows, deleted rows, sequentially scanned rows, and randomly scanned rows in the result set of the **SELECT** statement. Except that the number of rows returned by the result set is the same as the number of rows in the result set of the **SELECT** statement and is recorded only on the CN, the activity information of other rows is recorded on the DN. The statistical value reflects the row activities during the entire query execution process, including scanning and modifying related system tables, metadata tables, and data tables. The value of this parameter is related to the data volume and related parameter settings. That is, the statistical value is greater than or equal to the scanning and modification times of actual data tables.
- Time distribution, including DB_TIME/CPU_TIME/EXECUTION_TIME/PARSE_TIME/PLAN_TIME/REWRITE_TIME/PL_EXECUTION_TIME/PL_COMPILATION_TIME/NET_SEND_TIME/DATA_IO_TIME. For details, see [Table 18-120](#). The information is collected on both CNs and DNs and is displayed during view query.
- Number of soft and hard parsing times, such as the number of soft parsing times (cache plan) and hard parsing times (generation plan). If the cache plan is executed this time, the number of soft parsing times increases by 1. If the generation plan is regenerated this time, the number of hard parsing times increases by 1. This number is counted on both CNs and DNs and is displayed during view query.

The Unique SQL statistics function has the following restrictions:

- Detailed statistics are displayed only for successfully executed SQL statements. Otherwise, only query, node, and user information are recorded.
- Based on the Unique SQL ID generation algorithm, different IDs may be generated when the same SQL statement is executed on different nodes.
- If the Unique SQL statistics function is enabled, the CN collects statistics on all received queries, including tool and user queries.
- If an SQL statement contains multiple SQL statements or similar stored procedures, a Unique SQL statement is generated for the outermost SQL statement. The statistics of all sub-SQL statements are summarized to the Unique SQL record.
- The response time statistics of Unique SQL does not include the time of the **NET_SEND_TIME** phase. Therefore, there is no comparison between **EXECUTION_TIME** and **elapse_time**.

When a common user accesses the **GS_INSTR_UNIQUE_SQL** view, only the Unique SQL information about the user is displayed. When an administrator accesses the **GS_INSTR_UNIQUE_SQL** view, all Unique SQL information about the current node is displayed. The **GS_INSTR_UNIQUE_SQL** view can be queried on both CNs and DNs. The DN displays the Unique SQL statistics of the local node, and the CN displays the complete Unique SQL statistics of the local node. That is, the CN collects the Unique SQL execution information of the CN from other CNs and DNs and displays the information. You can query the **GS_INSTR_UNIQUE_SQL** view to locate the top SQL statements that consume

different resources, providing a basis for cluster performance optimization and maintenance.

Table 18-120 GS_INSTR_UNIQUE_SQL columns

Name	Type	Description
node_name	name	Name of the CN that receives SQL statements
node_id	integer	Node ID, which is the same as the value of node_id in the pgxc_node table
user_name	name	Username
user_id	oid	User ID
unique_sql_id	bigint	Normalized Unique SQL ID
query	text	Normalized SQL text
n_calls	bigint	Number of successful execution times
min_elapse_time	bigint	Minimum running time of the SQL statement in the database (unit: μ s)
max_elapse_time	bigint	Maximum running time of SQL statements in the database (unit: μ s)
total_elapse_time	bigint	Total running time of SQL statements in the database (unit: μ s)
n_returned_rows	bigint	Row activity - Number of rows in the result set returned by the SELECT statement
n_tuples_fetched	bigint	Row activity - Randomly scan rows (column-store tables/foreign tables are not counted.)
n_tuples_returned	bigint	Row activity - Sequential scan rows (Column-store tables/foreign tables are not counted.)
n_tuples_inserted	bigint	Row activity - Inserted rows

Name	Type	Description
n_tuples_updated	bigint	Row activity - Updated rows
n_tuples_deleted	bigint	Row activity - Deleted rows
n_blocks_fetched	bigint	Block access times of the buffer, that is, physical read/I/O
n_blocks_hit	bigint	Block hits of the buffer, that is, logical read/cache
n_soft_parse	bigint	Number of soft parsing times (cache plan)
n_hard_parse	bigint	Number of hard parsing times (generation plan)
db_time	bigint	Valid DB execution time, including the waiting time and network sending time. If multiple threads are involved in query execution, the value of DB_TIME is the sum of DB_TIME of multiple threads (unit: μ s).
cpu_time	bigint	CPU execution time, excluding the sleep time (unit: μ s)
execution_time	bigint	SQL execution time in the query executor, DDL statements, and statements (such as Copy statements) that are not executed by the executor are not counted (unit: μ s).
parse_time	bigint	SQL parsing time (unit: μ s)
plan_time	bigint	SQL generation plan time (unit: μ s)
rewrite_time	bigint	SQL rewriting time (unit: μ s)

Name	Type	Description
pl_execution_time	bigint	Execution time of the plpgsql procedural language function (unit: μ s)
pl_compilation_time	bigint	Compilation time of the plpgsql procedural language function (unit: μ s)
net_send_time	bigint	Network time, including the time spent by the CN in sending data to the client and the time spent by the DN in sending data to the CN (unit: μ s)
data_io_time	bigint	File I/O time (unit: μ s)

18.3.48 GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS displays load management information about CPU usage of ongoing complex jobs executed by the current user.

Table 18-121 GS_SESSION_CPU_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
username	name	Name of the user logging in to the backend
pid	bigint	ID of a backend process
start_time	timestamp with time zone	Time when the statement starts to be executed
min_cpu_time	bigint	Minimum CPU time of the statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of the statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of the statement across all DNs. The unit is ms.
query	text	Statement that is being executed
node_group	text	Logical cluster of the user running the statement

18.3.49 GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS displays load management information about memory usage of ongoing complex jobs executed by the current user.

Table 18-122 GS_SESSION_MEMORY_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
username	name	Name of the user logging in to the backend
pid	bigint	ID of a backend process
start_time	timestamp with time zone	Time when the statement starts to be executed
min_peak_memory	integer	Minimum memory peak of a statement across all DNs, in MB
max_peak_memory	integer	Maximum memory peak of a statement across all DNs, in MB
spill_info	text	Information about statement flushing into disks on DNs None indicates that the statement has not been flushed to disks on any DNs. All indicates that the statement has been flushed to disks on every DN. [a:b] indicates that the statement has been flushed to disks on <i>a</i> of <i>b</i> DNs.
query	text	Statement that is being executed
node_group	text	Logical cluster of the user running the statement

18.3.50 GS_SQL_COUNT

GS_SQL_COUNT displays statistics about the five types of statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) executed on the current node of the database, including the number of execution times, response time (the maximum, minimum, average, and total response time of the other four types of statements except the **MERGE INTO** statement, in microseconds), and the number of execution times of **DDL**, **DML**, and **DCL statements**.

The classification of **DDL**, **DML**, and **DCL** statements in the **GS_SQL_COUNT** view is slightly different from that of the SQL syntax. The details are as follows:

- User-related statements, such as **CREATE/ALTER/DROP USER** and **CREATE/ALTER/DROP ROLE**, are of the DCL type.
- Transaction-related statements such as **BEGIN/COMMIT/SET CONSTRAINTS/ROLLBACK/SAVEPOINT/START** are of the DCL type.
- **ALTER SYSTEM KILL SESSION** is equivalent to the **SELECT pg_terminate_backend()** statement and is of the DML type.

The classification of other statements is similar to the definition in the SQL syntax.

When a common user queries the **GS_SQL_COUNT** view, only the statistics of the current node of the user can be viewed. When an administrator queries the **GS_SQL_COUNT** view, the statistics of the current node of all users can be viewed. 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 18-123 GS_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name
user_name	name	User 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
mergeinto_count	bigint	Number of MERGE INTO statements
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
avg_select_elapse	bigint	Average response time of SELECT statements
max_select_elapse	bigint	Maximum response time of SELECT statements
min_select_elapse	bigint	Minimum response time of SELECT statements
total_update_elapse	bigint	Total response time of UPDATE statements
avg_update_elapse	bigint	Average response time of UPDATE statements

Name	Type	Description
max_update_elapsed	bigint	Maximum response time of UPDATE statements
min_update_elapsed	bigint	Minimum response time of UPDATE statements
total_delete_elapsed	bigint	Total response time of DELETE statements
avg_delete_elapsed	bigint	Average response time of DELETE statements
max_delete_elapsed	bigint	Maximum response time of DELETE statements
min_delete_elapsed	bigint	Minimum response time of DELETE statements
total_insert_elapsed	bigint	Total response time of INSERT statements
avg_insert_elapsed	bigint	Average response time of INSERT statements
max_insert_elapsed	bigint	Maximum response time of INSERT statements
min_insert_elapsed	bigint	Minimum response time of INSERT statements

18.3.51 GS_WLM_CGROUP_INFO

GS_WLM_CGROUP_INFO displays information about a Cgroup for a job that is being executed.

Table 18-124 GS_WLM_CGROUP_INFO columns

Name	Type	Description
cgoup_name	text	Name of a Cgroup
priority	interger	Priority of a job
usage_pencent	interger	Percentage of resources used by a Cgroup
shares	bigint	CPU quota allocated to a Cgroup
cpuacct	bigint	Allocated CPU quota
cpuset	text	Allocated CPU cores
relpath	text	Relative path of a Cgroup

Name	Type	Description
valid	text	Whether a Cgroup is valid
node_group	text	Name of a logical cluster

18.3.52 GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY displays the records of operators in jobs that have been executed by the current user on the current CN.

This view is used by Database Manager to query data from the kernel. Data in the kernel is cleared every 3 minutes.

18.3.53 GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS displays the operators of the jobs that are being executed by the current user.

Table 18-125 GS_WLM_OPERATOR_STATISTICS columns

Name	Type	Description
queryid	bigint	Internal query_id used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	plan_node_id of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
status	text	Execution status of the current operator. Its value can be finished or running .
query_dop	integer	DOP of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the current operator
min_peak_memory	integer	Minimum peak memory used by the current operator on all DN. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DN. The unit is MB.

Name	Type	Description
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 .
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 .
average_spill_size	integer	Average spilled data of all DNs when a spill occurs. The unit is MB. The default value is 0 .
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
warning	text	Warning. The following warnings are displayed: 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

18.3.54 GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY displays load management information about a completed job executed by the current user on the current CN. This view is used by Database Manager to query data from GaussDB(DWS). Data in the GaussDB(DWS) is cleared every 3 minutes.

Table 18-126 GS_WLM_SESSION_HISTORY columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
dbname	text	Name of the database the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the CN where the statement is run
username	text	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend
client_addr	inet	IP address of the client connected to this 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 only be non-null 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, or -1 if a Unix socket is used
query_band	text	Job type, which is specified by the query_band parameter. The default value is a null string.
block_time	bigint	Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms.
start_time	timestamp with time zone	Time when the statement starts to be run
finish_time	timestamp with time zone	Time when the statement execution ends
duration	bigint	Execution time of a statement. The unit is ms.
estimate_total_time	bigint	Estimated execution time of a statement. The unit is ms.
status	text	Final statement execution status. Its value can be finished (normal) or aborted (abnormal).

Name	Type	Description
abort_info	text	Exception information displayed if the final statement execution status is aborted .
resource_pool	text	Resource pool used by the user
control_group	text	Cgroup used by the statement
min_peak_memory	integer	Minimum memory peak of a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. None indicates that the statement has not been flushed to disks on any DNs. All indicates that the statement has been flushed to disks on every DN. [<i>a</i>:<i>b</i>] indicates that the statement has been flushed to disks on <i>a</i> of <i>b</i> DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 .
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 .
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 .
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dn_time_skew_percent	integer	Execution time skew of a statement among DNs.

Name	Type	Description
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew across DNs.
warning	text	Warning. The following warnings and warnings related to Optimizing SQL Self-Diagnosis are displayed: 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement executed
query_plan	text	Execution plan of a statement
node_group	text	Logical cluster of the user running the statement

18.3.55 GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS displays load management information about jobs being executed by the current user on the current CN.

Table 18-127 GS_WLM_SESSION_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
dbname	name	Name of the database the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the CN where the statement is executed
username	name	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend
client_addr	inet	IP address of the client connected to this 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 only be non-null 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, or -1 if a Unix socket is used
query_band	text	Job type, which is specified by the GUC parameter query_band parameter. The default value is a null string.
pid	bigint	Process ID of the backend
block_time	bigint	Block time before the statement is executed. The unit is ms.
start_time	timestamp with time zone	Time when the statement starts to be executed
duration	bigint	For how long a statement has been executing. The unit is ms.

Name	Type	Description
estimate_total_time	bigint	Estimated execution time of a statement. The unit is ms.
estimate_left_time	bigint	Estimated remaining time of statement execution. The unit is ms.
enqueue	text	Workload management resource status
resource_pool	name	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory used by the statement. The unit is MB.
min_peak_memory	integer	Minimum memory peak of a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. None indicates that the statement has not been flushed to disks on any DNs. All indicates that the statement has been flushed to disks on every DN. [a:b] indicates that the statement has been flushed to disks on <i>a</i> of <i>b</i> DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.

Name	Type	Description
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dn_time_skew_percent	bigint	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew across DNs.
warning	text	Warning. The following warnings and warnings related to Optimizing SQL Self-Diagnosis are displayed: 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement that is being executed

Name	Type	Description
query_plan	text	Execution plan of a statement
node_group	text	Logical cluster of the user running the statement

18.3.56 GS_WLM_WORKLOAD_RECORDS

GS_WLM_WORKLOAD_RECORDS displays the status of job executed by the current user on CNs.

Table 18-128 GS_WLM_WORKLOAD_RECORDS columns

Name	Type	Description
node_name	text	Name of the CN where a job is executed
thread_id	bigint	ID of a backend process
processid	integer	PID of a backend thread
time_stamp	name	Time when a statement starts to be executed
username	interger	Name of a user logged in to the backend
memory	interger	Memory required by a statement
active_points	interger	Number of resources consumed by a statement in a resource pool
priority	interger	Priority of a job
resource_pool	text	Resource pool to which a job belongs
status	text	Job execution status. Its value can be: • pending • running • finished • aborted • Unknown
control_group	text	Cgroups used by a job
enqueue	text	Queue that a job is in. Its value can be: • GLOBAL: global queue • RESPOOL: resource pool queue • Active: not in a queue
query	text	Statement that is being executed
node_group	text	Name of a logical cluster

18.3.57 PGXC_WORKLOAD_SQL_COUNT

PGXC_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on all CNs in a cluster, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements. It is accessible only to users with system administrator rights.

Table 18-129 PGXC_WORKLOAD_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload Cgroup 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

18.3.58 PGXC_WORKLOAD_SQL_ELAPSE_TIME

PGXC_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on all CNs in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond. It is accessible only to users with system administrator rights.

Table 18-130 PGXC_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload Cgroup name

Name	Type	Description
total_select_elapse	bigint	Total response time of SELECT statements
max_select_elapse	bigint	Maximum response time of SELECT statements
min_select_elapse	bigint	Minimum response time of SELECT statements
avg_select_elapse	bigint	Average response time of SELECT statements
total_update_elapse	bigint	Total response time of UPDATE statements
max_update_elapse	bigint	Maximum response time of UPDATE statements
min_update_elapse	bigint	Minimum response time of UPDATE statements
avg_update_elapse	bigint	Average response time of UPDATE statements
total_insert_elapse	bigint	Total response time of INSERT statements
max_insert_elapse	bigint	Maximum response time of INSERT statements
min_insert_elapse	bigint	Minimum response time of INSERT statements
avg_insert_elapse	bigint	Average response time of INSERT statements
total_delete_elapse	bigint	Total response time of DELETE statements
max_delete_elapse	bigint	Maximum response time of DELETE statements
min_delete_elapse	bigint	Minimum response time of DELETE statements
avg_delete_elapse	bigint	Average response time of DELETE statements

18.3.59 GS_WORKLOAD_TRANSACTION

GS_WORKLOAD_TRANSACTION provides transaction information about workload cgroups on a single CN. The database records the number of times that each workload Cgroup commits and rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 18-131 GS_WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload Cgroup name
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time
resp_max	bigint	Maximum response time
resp_avg	bigint	Average response time
resp_total	bigint	Total response time

18.3.60 GS_STAT_DB_CU

GS_STAT_DB_CU displays CU hits in a database and in each node in a cluster. You can clear it using **gs_stat_reset()**.

Table 18-132 GS_STAT_DB_CU columns

Name	Type	Description
node_name1	text	Node name
db_name	text	Database 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

18.3.61 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU displays the CU hit rate of running sessions on each node in a cluster. This data about a session is cleared when you exit this session or restart the cluster.

Table 18-133 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 hard disk synchronous reads
hdd_asyn_read	integer	Number of hard disk asynchronous reads

18.3.62 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL displays statistics about memory usage of the logical cluster that the current database belongs to in the unit of MB.

Table 18-134 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Name of a logical cluster
memorytype	text	Memory type. Its value can be: • ng_total_memory: total memory of the logical cluster • ng_used_memory: memory usage of the logical cluster • ng_estimate_memory: estimated memory usage of the logical cluster • ng_foreignrp_memsize: total memory of the external resource pool of the logical cluster • ng_foreignrp_memsize: memory usage of the external resource pool of the logical cluster • ng_foreignrp_peaksize: peak memory usage of the external resource pool of the logical cluster • ng_foreignrp_mempct: percentage of the external resource pool of the logical cluster to the total memory of the logical cluster • ng_foreignrp_estmsize: estimated memory usage of the external resource pool of the logical cluster
memorybytes	integer	Size of allocated memory-typed memory

18.3.63 GS_USER_TRANSACTION

GS_USER_TRANSACTION provides transaction information about users on a single CN. The database records the number of times that each user commits and

rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 18-135 GS_USER_TRANSACTION columns

Name	Type	Description
username	name	Username
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time
resp_max	bigint	Maximum response time
resp_avg	bigint	Average response time
resp_total	bigint	Total response time

18.3.64 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS displays the extension versions of certain database features.

Table 18-136 PG_AVAILABLE_EXTENSION_VERSIONS columns

Name	Type	Description
name	name	Extension name
version	text	Version name
installed	Boolean	The value is true if the version of this extension is currently installed.
superuser	Boolean	The value is true if only system administrators are allowed to install this extension.
relocatable	Boolean	The value is true if an extension can be relocated to another schema.
schema	name	Name of the schema that the extension must be installed into. The value is null if the extension is partially or fully relocatable.
requires	name[]	Names of prerequisite extensions. The value is null if there are no prerequisite extensions.
comment	text	Comment string from the extension's control file

18.3.65 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS displays the extended information about certain database features.

Table 18-137 PG_AVAILABLE_EXTENSIONS columns

Name	Type	Description
name	name	Extension name
default_version	text	Name of default version. The value is NULL if none is specified.
installed_version	text	Currently installed version of the extension. The value is NULL if no version is installed.
comment	text	Comment string from the extension's control file

18.3.66 PG_COMM_CLIENT_INFO

PG_COMM_CLIENT_INFO stores the client connection information of a single node. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 18-138 PG_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	Client application name
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	It 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	If the connection is a logical connection, sid is displayed. If -1 is displayed, the current connection is a physical connection.

18.3.67 PG_COMM_DELAY

PG_COMM_DELAY displays the communication library delay status for a single DN.

Table 18-139 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
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. Its unit is 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. The unit is microsecond.
max_delay	integer	Maximum delay of the current physical connection within 1 minute. The unit is microsecond.

18.3.68 PG_COMM_STATUS

PG_COMM_STATUS displays the communication library status for a single DN.

Table 18-140 PG_COMM_STATUS columns

Name	Type	Description
node_name	text	Specifies the node name.
rxpck/s	integer	Receiving rate of the communication library on a node. The unit is byte/s.
txpck/s	integer	Sending rate of the communication library on a node. The unit is byte/s.
rxkB/s	bigint	Receiving rate of the communication library on a node. The unit is KB/s.
txkB/s	bigint	Sending rate of the communication library on a node. The unit is KB/s.

Name	Type	Description
buffer	bigint	Size of the buffer of the Cmailbox.
memKB(libcomm)	bigint	Communication memory size of the libcomm process, in bytes.
memKB(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.

18.3.69 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 18-141 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	Current status of the stream
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

Name	Type	Description
recv_smp	integer	smpid of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received from the stream. The unit is byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average receiving rate of the stream. The unit is byte/s.
quota	bigint	Current communication quota value of the stream. The unit is byte.
buff_usize	bigint	Current size of the data cache of the stream. The unit is byte.

18.3.70 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 18-142 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	Current status of the stream
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

Name	Type	Description
recv_smp	integer	smpid of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream. The unit is Byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average sending rate of the stream. The unit is Byte/s.
quota	bigint	Current communication quota value of the stream. The unit is Byte.
wait_quota	bigint	Extra time generated when the stream waits the quota value. The unit is ms.

18.3.71 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG displays the Cgroup configuration information in the system.

Table 18-143 PG_CONTROL_GROUP_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the cgroup

18.3.72 PG_CURSORS

PG_CURSORS displays the cursors that are currently available.

Table 18-144 PG_CURSORS columns

Name	Type	Description
name	text	Cursor name
statement	text	Query statement when the cursor is declared to change
is_holdable	Boolean	Whether the cursor is holdable (that is, it can be accessed after the transaction that declared the cursor has committed). If it is, its value is true .
is_binary	Boolean	Whether the cursor was declared BINARY. If it was, its value is true .

Name	Type	Description
is_scrollable	Boolean	Whether the cursor is scrollable (that is, it allows rows to be retrieved in a nonsequential manner). If it is, its value is true .
creation_time	timestamp with time zone	Timestamp at which the cursor is declared

18.3.73 PG_EXT_STATS

PG_EXT_STATS displays extension statistics stored in the **PG_STATISTIC_EXT** table. The extension statistics means multiple columns of statistics.

Table 18-145 PG_EXT_STATS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESP ACE .nspname	Name of the schema that contains a table
tablename	name	PG_CLASS .rel name	Name of a table
attname	int2vector	PG_STATISTI C_EXT .stakey	Indicates the columns to be combined for collecting statistics.
inherited	Boolean	-	Includes inherited sub-columns if the value is true ; otherwise, indicates the column in a specified table.
null_frac	real	-	Percentage of column combinations that are null to all records
avg_width	integer	-	Average width of column combinations. The unit is byte.

Name	Type	Reference	Description
n_distinct	real	-	- Estimated number of distinct values in a column combination 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 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. For example, -1 indicates that the number of distinct values is the same as the number of rows for a column combination. - The number of distinct values is unknown if the value is 0.
n_dndistinct	real	-	Number of unique not-null data values in the dn1 column combination - 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 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, most_common_vals_null will be NULL. None of the most common values in most_common_vals is NULL.

Name	Type	Reference	Description
most_common_freqs	real[]	-	List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals 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, most_common_vals_null will be NULL. At least one of the common values in most_common_vals_null is NULL.
most_common_freqs_null	real[]	-	List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals_null is NULL)

18.3.74 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS displays the information about backend threads on the CN that are connected to the current standby DN.

Table 18-146 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 performed by the backend thread

18.3.75 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME displays the catchup information of the currently active primary/standby instance sending thread on a single DN.

Table 18-147 PG_GET_SENDERS_CATCHUP_TIME columns

Name	Type	Description
pid	bigint	Current sender thread ID
lwpid	integer	Current sender lwpid
local_role	text	Local role
peer_role	text	Peer role
state	text	Current sender's replication status
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 a catchup task

18.3.76 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between roles.

Table 18-148 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

18.3.77 PG_INDEXES

PG_INDEXES displays access to useful information about each index in the database.

Table 18-149 PG_INDEXES columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema that contains tables and indexes
tablename	name	PG_CLASS.relname	Name of the table for which the index serves
indexname	name	PG_CLASS.relname	Index name
tablespace	name	PG_TABLESPACE.spcname	Name of the tablespace that contains the index
indexdef	text	-	Index definition (a reconstructed CREATE INDEX command)

18.3.78 PG_LOCKS

PG_LOCKS displays information about the locks held by open transactions.

Table 18-150 PG_LOCKS columns

Name	Type	Reference	Description
locktype	text	-	Type of the locked object: relation, extend, page, tuple, transactionid, virtualxid, object, userlock, and advisory
database	oid	PG_DATABASE.oid	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 target is a transaction.
relation	oid	PG_CLASS.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. If the object is not a relation page or row page, the value is NULL .
tuple	smallint	-	Row number targeted by the lock within the page. If the object is not a row, the value is NULL .

Name	Type	Reference	Description
virtualxid	text	-	Virtual ID of the transaction targeted by the lock. If the object is not a virtual transaction ID, the value is NULL .
transactionid	xid	-	ID of the transaction targeted by the lock. If the object is not a transaction ID, the value is NULL .
classid	oid	PG_CLASS .oid	OID of the system table that contains the object. If the object is not a general database object, the value is NULL .
objid	oid	-	OID of the lock target within its system table. If the target is not a general database object, the value is NULL .
objsubid	smallint	-	Column number for a column in the table. The value is 0 if the target is some other object type. If the object is not a general database object, the value is NULL .
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. This is NULL if the lock is held by a prepared transaction.
mode	text	-	Lock mode held or desired by this thread
granted	Boolean	-	Whether the lock is a held lock • The value is true if the lock is a held lock. • The value is false if the lock is an awaited lock.
fastpath	Boolean	-	Whether the lock is obtained through fast-path (true) or main lock table (false)

18.3.79 PG_NODE_ENV

PG_NODE_ENV displays the environmental variable information about the current node.

Table 18-151 PG_NODE_ENV columns

Name	Type	Description
node_name	text	Name of the current node
host	text	Host name of the current node
process	integer	Process ID of the current node
port	integer	Port ID of the current node
installpath	text	Installation directory of current node
datapath	text	Data directory of the current node
log_directory	text	Log directory of the current node

18.3.80 PG_OS_THREADS

PG_OS_THREADS displays the status information about all the threads under the current node.

Table 18-152 PG_OS_THREADS columns

Name	Type	Description
node_name	text	Name of the current node
pid	bigint	Thread number 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

18.3.81 PG_POOLER_STATUS

PG_POOLER_STATUS displays the cache connection status in the pooler.
PG_POOLER_STATUS can only query on the CN, and displays the connection cache information about the pooler module.

Table 18-153 PG_POOLER_STATUS columns

Name	Type	Description
database	text	Database name

Name	Type	Description
user_name	text	User name
tid	bigint	ID of a thread 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 in use • t (true): indicates that the connection is in use. • f (false): indicates that the connection is not in use.
fdsock	bigint	Peer socket.
remote_pid	bigint	Peer thread ID.
session_params	text	GUC session parameter delivered by the connection.

18.3.82 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 18-154 PG_PREPARED_STATEMENTS columns

Name	Type	Description
name	text	Identifier of the prepared statement
statement	text	Query string for creating this prepared statement For prepared statements created through SQL, this is the PREPARE statement submitted 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 casting the regtype value to oid.

Name	Type	Description
from_sql	Boolean	How a prepared statement was created • true: The prepared statement was created through the PREPARE statement. • false The statement was prepared through the frontend/backend protocol.

18.3.83 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 18-155 PG_PREPARED_XACTS columns

Name	Type	Reference	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	PG_AUTHID .rolname	Name of the user that executes the transaction
database	name	PG_DATABASE .dbname	Name of the database in which the transaction is executed

18.3.84 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS displays the replication node information.

Table 18-156 PG_REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Name of a replication node
plugin	name	Name of the output plug-in of the logical replication slot

Name	Type	Description
slot_type	text	Type of a replication node
datoid	oid	OID of the database on the replication node
database	name	Name of the database on the replication node
active	Boolean	Whether the replication node is active
xmin	xid	Transaction ID of the replication node
catalog_xmin	text	ID of the earliest-decoded transaction corresponding to the logical replication slot
restart_lsn	text	Xlog file information on the replication node
dummy_standby	Boolean	Whether the replication node is the dummy standby node

18.3.85 PG_ROLES

PG_ROLES displays information about database roles.

Table 18-157 PG_ROLES columns

Name	Type	Reference	Description
rolname	name	-	Role name
rolsuper	Boolean	-	Whether the role is the initial system administrator with the highest permission
rolinherit	Boolean	-	Whether the role inherits permissions for this type of roles
rolcreaterole	Boolean	-	Whether the role can create other roles
rolcreatedb	Boolean	-	Whether the role can create databases
rolcatupdate	Boolean	-	Whether the role can update system tables directly. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users.
rolcanlogin	Boolean	-	Whether the role can log in to the database
rolreplication	Boolean	-	Whether the role can be replicated

Name	Type	Reference	Description
rolauditadmin	Boolean	-	Whether the role is an audit system administrator
rolsystemadmin	Boolean	-	Whether the role is a system administrator
rolconnlimit	integer	-	Sets the maximum number of concurrent connections this role can make if this role can log in. -1 indicates no limit.
rolpassword	text	-	Not the password (always reads as *****)
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
rolrespool	name	-	Resource pool that a user can use
rolparentid	oid	PG_AUTHID.rolparentid	OID of a group user to which the user belongs
roltabspace	text	-	The storage space of the user permanent table.
roltemp space	text	-	The storage space of the user temporary table.
rolspillspace	text	-	The operator disk flushing space of the user.
rolconfig	text[]	-	Session defaults for runtime configuration variables
oid	oid	PG_AUTHID.oid	ID of the role
roluseft	Boolean	PG_AUTHID.roluseft	Whether the role can perform operations on foreign tables
nodegroup	name	-	Name of the logical cluster associated with the role. If no logical cluster is associated, this column is left empty.

18.3.86 PG_RULES

PG_RULES displays information about rewrite rules.

Table 18-158 PG_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
tablename	name	Name of the table the rule is for
rulename	name	Rule name
definition	text	Rule definition (a reconstructed creation command)

18.3.87 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays the running transaction information on the current node.

Table 18-159 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 xmin on the node
vacuum	Boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the prepared status. 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

18.3.88 PG_SECLABELS

PG_SECLABELS displays information about security labels.

Table 18-160 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	PG_CLASS .oid	OID of the system table that contains the object
objsubid	integer	-	For a security label on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is 0 .
objtype	text	-	Type of the object to which this label applies
objnamespace	oid	PG_NAMESPACE .oid	OID of the namespace for this object, if applicable; otherwise NULL.
objname	text	-	Name of the object to which the label applies
provider	text	PG_SECLABEL .provider	Label provider associated with this label
label	text	PG_SECLABEL .label	Security label applied to this object

18.3.89 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT displays the corresponding load management information about the task currently executed by the user.

Table 18-161 PG_SESSION_WLMSTAT columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
datname	name	Name of the database the backend is connected to
threadid	bigint	Process ID of the backend
processid	integer	Thread PID of the backend
usesysid	oid	OID of the user who logged into the backend
appname	text	Name of the application that is connected to the backend

Name	Type	Description
username	name	Name of the user logged in to the backend
priority	bigint	Priority of Cgroup where the statement is located
attribute	text	Statement attributes • Ordinary: default attribute of a statement before it is parsed by the database • Simple: simple statements • Complicated: complicated statements • Internal: internal statement of the database
block_time	bigint	Pending duration of the statements by now (unit: s)
elapsed_time	bigint	Actual execution duration of the statements by now (unit: s)
total_cpu_time	bigint	Total CPU usage duration of the statement on the DN in the last period (unit: s)
cpu_skew_percent	integer	CPU usage inclination ratio of the statement on the DN in the last period
statement_mem	integer	statement_mem used for executing the statement (reserved column)
active_points	integer	Number of concurrently active points occupied by the statement in the resource pool
dop_value	integer	DOP value obtained by the statement from the resource pool
control_group	text	Cgroup currently used by the statement
status	text	Status of a statement, including: • pending: waiting to be executed • running: being executed • finished: finished normally (If enqueue is set to StoredProc or Transaction, this state indicates that only a part of jobs in the statement have been executed. This state persists until the finish of this statement.) • aborted: terminated unexpectedly • active: normal status except for those above • unknown: unknown status

Name	Type	Description
enqueue	text	Current queuing status of the statements, including: • Global: queuing in the global queue • Respool: queuing in the resource pool queue • CentralQueue: queuing on the CCN • Transaction: being in a transaction block • StoredProc: being in a stored procedure • None: not in a queue • Forced None: being forcibly executed (transaction block statement or stored procedure statement are) because the statement waiting time exceeds the specified value
resource_pool	name	Current resource pool where the statements are located.
query	text	Text of this backend's most recent query. If state is active , this column shows the executing query. In all other states, it shows the last query that was executed.
isplana	bool	In logical cluster mode, indicates whether a statement occupies the resources of other logical clusters. The default value is f (does not occupy).
node_group	text	Logical cluster of the user running the statement

18.3.90 PG_SESSION_IOSTAT

PG_SESSION_IOSTAT displays the I/O load management information about the task currently executed by the user.

IOPS is counted by ones for column storage and by thousands for row storage.

Table 18-162 PG_SESSION_IOSTAT columns

Name	Type	Description
query_id	bigint	Job ID
mincurriops	integer	Minimum I/O of the current job across DNs
maxcurriops	integer	Maximum I/O of the current job across DNs
minpeaklops	integer	Minimum peak I/O of the current job across DNs
maxpeaklops	integer	Maximum peak I/O of the current job across DNs

Name	Type	Description
io_limits	integer	io_limits set for the job
io_priority	text	io_priority set for the job
query	text	Job
node_group	text	Logical cluster of the user running the job

18.3.91 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 18-163 PG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current value of the parameter
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 of parameter values including internal, backend, superuser, and user
vartype	text	Parameter type. It can be 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 data, the value of this column is null.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric data, the value of this column is null.
enumvals	text[]	Valid values of an enum-typed 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

Name	Type	Description
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.

18.3.92 PG_SHADOW

PG_SHADOW displays properties of all roles that are marked as **rolcanlogin** in **PG_AUTHID**.

The name stems from the fact that this table should not be readable by the public since it contains passwords. **PG_USER** is a publicly readable view on **PG_SHADOW** that blanks out the password column.

Table 18-164 PG_SHADOW columns

Name	Type	Reference	Description
username	name	PG_AUTHID .rolname	User name
usesysid	oid	PG_AUTHID .oid	ID of a user
usecreatedb	Boolean	-	Indicates that the user can create databases.
usesuper	Boolean	-	Indicates that the user is an administrator.
usecatupd	Boolean	-	Indicates that the user can update system catalogs. Even the system administrator cannot do this unless this column is true .
userepl	Boolean	-	User can initiate streaming replication and put the system in and out of backup mode.
passwd	text	-	Password (possibly encrypted); null if none. See PG_AUTHID for details about how encrypted passwords are stored.

Name	Type	Reference	Description
valbegin	timestamp with time zone	-	Account validity start time; null if no start time
valuntil	timestamp with time zone	-	Password expiry time; null if no expiration
respool	name	-	Resource pool used by the user
parent	oid	-	Parent resource pool
spacelimit	text	-	The storage space of the permanent table.
tempspacelimit	text	-	The storage space of the temporary table.
spillspacelimit	text	-	The operator disk flushing space.
useconfig	text[]	-	Session defaults for runtime configuration variables

18.3.93 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL displays usage information about all the shared memory contexts.

Table 18-165 PG_SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the context in the memory
level	smallint	Hierarchy of the memory context
parent	text	Context of the parent memory
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

18.3.94 PG_STATS

PG_STATS displays the single-column statistics stored in the **pg_statistic** table.

Table 18-166 PG_STATS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESP ACE .nspname	Name of the schema that contains the table
tablename	name	PG_CLASS .rel name	Name of the table
attname	name	PG_ATTRIBU TE .attname	Column name
inherited	Boolean	-	Includes inherited sub-columns if the value is true ; otherwise, indicates the column in a specified table.
null_frac	real	-	Percentage of column entries that are null
avg_width	integer	-	Average width in bytes of column's entries
n_distinct	real	-	- Estimated number of distinct values in the column 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 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. For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.

Name	Type	Reference	Description
n_dndistinct	real	-	Number of unique non-null data 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=-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 combination does not have the most common values, it will be NULL .
most_common_freqs	real[]	-	List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL)
histogram_bounds	anyarray	-	List of values that divide the column's values into groups of equal proportion. The values in most_common_vals , if present, are omitted from this histogram calculation. This field is null if the field data type does not have a < operator or if the most_common_vals list accounts for the entire population.
correlation	real	-	Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator.
most_common_elems	anyarray	-	Specifies a list of non-null element values most often appearing.

Name	Type	Reference	Description
most_common_elem_freqs	real[]	-	Specifies a list of the frequencies of the most common element values.
elem_count_histogram	real[]	-	Specifies a histogram of the counts of distinct non-null element values.

18.3.95 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY displays information about the current user's queries.

Table 18-167 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	Process ID of the backend
usesysid	oid	OID of the user logging in to the backend
username	name	OID of the user logging in to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either 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 only be non-null 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, or -1 if a Unix socket is used

Name	Type	Description
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server.
xact_start	timestamp with time zone	Time when the current transaction was started, or NULL if no transaction is active. If the current query is the first of its transaction, this column is equal to the query_start column.
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 for the last status change
waiting	Boolean	Whether the backend is currently waiting on a lock. If it is, its value is true .
enqueue	text	Queuing status of a statement. Its value can be: • waiting in queue: The statement is in the queue. • waiting in global queue: The statement is in the global queue. • waiting in respool queue: The statement is in the resource pool queue. • waiting in ccn queue: The job is in the CCN queue. • Empty: The statement is running.

Name	Type	Description
state	text	Current overall state of this backend. Its value can be: • active: The backend is executing queries. • idle: The backend is waiting for new client commands. • 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 only their own session status. 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 pg_stat_activity is empty. <pre>SELECT datname, username, usesysid, state, pid FROM pg_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	Text of this backend's most recent query. If state is active , this column shows the executing query. In all other states, it shows the last query that was executed.
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database (for details, see connection_info)

18.3.96 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES displays access information about all indexes in the database, with information about each index displayed in a row.

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, a 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 18-168 PG_STAT_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 this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_scan	bigint	Number of index scans initiated on this index
idx_tup_read	bigint	Number of index entries returned by scans on this index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using this index

18.3.97 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES displays access information about all rows in all tables (including TOAST tables) in the database.

Table 18-169 PG_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of the table
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by HOT (no separate index update is 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 automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was automatically vacuumed
vacuum_count	bigint	Number of vacuum operations (excluding VACUUM FULL)

Name	Type	Description
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations
last_data_changed	timestamp with time zone	Last time at which this table was updated (by INSERT/UPDATE/DELETE or EXCHANGE/TRUNCATE/DROP <i>partition</i>). This column is recorded only on the local CN.

18.3.98 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK displays statistics about page or CU verification failures after a node is started.

Table 18-170 PG_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
forknum	integer	File type
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first occurrence
last_time	timestamp with time zone	Time of the latest occurrence

18.3.99 PG_STAT_BGWRITER

PG_STAT_BGWRITER displays statistics about the background writer process's activity.

Table 18-171 PG_STAT_BGWRITER 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 amount of time that has been spent in the portion of checkpoint processing where files are written to disk, in milliseconds
checkpoint_sync_time	double precision	Total amount of time that has been spent in the portion of checkpoint processing where files are synchronized to disk, in milliseconds
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by a backend
buffers_backend_fsync	bigint	Number of times that a backend has to execute fsync
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were reset

18.3.100 PG_STAT_DATABASE

PG_STAT_DATABASE displays statistics about databases in the cluster, with statistics about each database displayed in a row.

Table 18-172 PG_STAT_DATABASE columns

Name	Type	Description
datid	oid	Database OID
datname	name	Database name

Name	Type	Description
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting 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 already in the buffer cache, so that a read was not necessary (this only includes hits in the PostgreSQL buffer cache, not the operating system's file system 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
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 conflicts that occur only on standby servers. For details, see PG_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 reading data file blocks by backends in this database, in milliseconds

Name	Type	Description
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database, in milliseconds
stats_reset	timestamp with time zone	Time at which these statistics were reset

18.3.101 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 18-173 PG_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database OID
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

18.3.102 PG_STAT_GET_MEM_MBYTES_RESERVED

PG_STAT_GET_MEM_MBYTES_RESERVED displays the current activity information of a thread stored in memory. You need to specify the thread ID (pid in [PG_STAT_ACTIVITY](#)) for query. If the thread ID is set to **0**, the current thread ID is used. For example:

```
SELECT pg_stat_get_mem_mbytes_reserved(0);
```

Table 18-174 PG_STAT_GET_MEM_MBYTES_RESERVED columns

Parameter	Description
ConnectInfo	Connection information
ParctlManager	Concurrency management information

Parameter	Description
GeneralParams	Basic parameter information
GeneralParams RPDATA	Basic resource pool information
ExceptionManager	Exception management information
CollectInfo	Collection information
GeneralInfo	Basic information
ParctlState	Concurrency status information
CPU INFO	CPU information
ControlGroup	Cgroup information
IOWSTATE	I/O status information

18.3.103 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS displays user-defined function status information in the namespace. (The language of the function is non-internal language.)

Table 18-175 PG_STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times this 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 itself, excluding other functions called by it

18.3.104 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 18-176 PG_STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans
idx_tup_read	bigint	Number of index entries returned by scans on this index
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans

18.3.105 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays status information about user-defined ordinary tables and TOAST tables in all namespaces.

Table 18-177 PG_STAT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data 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 d	bigint	Number of rows updated by HOT (no separate index update is 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 (excluding VACUUM FULL)
last_autovacuum	timestamp with time zone	Last time at which this table was automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was automatically analyzed
vacuum_count	bigint	Number of vacuum operations (excluding VACUUM FULL)
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations

18.3.106 PG_STAT_REPLICATION

PG_STAT_REPLICATION displays information about log synchronization status, such as the locations of the sender sending logs and the receiver receiving logs.

Table 18-178 PG_STAT_REPLICATION columns

Name	Type	Description
pid	bigint	PID of the thread
usesysid	oid	User system ID
username	name	Username
application_name	text	Application name
client_addr	inet	Client address
client_hostname	text	Client name

Name	Type	Description
client_port	integer	Client port number
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state (catch-up or consistent streaming)
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 duplication, synchronous duplication, or potential synchronization)

18.3.107 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays the index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas.

Table 18-179 PG_STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans
idx_tup_read	bigint	Number of index entries returned by scans on this index

Name	Type	Description
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans

18.3.108 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES displays the statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas.

Table 18-180 PG_STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data 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 (that is, 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 automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed

Name	Type	Description
last_autoanalyze	timestamp with time zone	Last time at which this table was automatically analyzed
vacuum_count	bigint	Number of vacuum operations (excluding VACUUM FULL)
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations

18.3.109 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in the namespaces.

Table 18-181 PG_STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started 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 (that is, with no separate index update required)

18.3.110 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays the transaction status information of the system catalog in the namespace.

Table 18-182 PG_STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started 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 d	bigint	Number of rows HOT updated (that is, with no separate index update required)

18.3.111 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS displays statistics about function executions, with statistics about each execution displayed in a row.

Table 18-183 PG_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times this function has been called
total_time	double precision	Total time spent in this function and all other functions called by it

Name	Type	Description
self_time	double precision	Total time spent in this function itself, excluding other functions called by it

18.3.112 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays the transaction status information of the user table in the namespace.

Table 18-184 PG_STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started 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 d	bigint	Number of rows HOT updated (that is, with no separate index update required)

18.3.113 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES contains each row of each index in the current database, showing I/O statistics about accesses to that specific index.

Table 18-185 PG_STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID

Name	Type	Description
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this index
idx_blks_hit	bigint	Number of buffer hits in this index

18.3.114 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES contains each row of each sequence in the current database, showing I/O statistics about accesses to that specific sequence.

Table 18-186 PG_STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema this sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from this sequence
blks_hit	bigint	Number of buffer hits in this sequence

18.3.115 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES contains one row for each table in the current database (including TOAST tables), showing I/O statistics about accesses to that specific table.

Table 18-187 PG_STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table

Name	Type	Description
idx_blks_read	bigint	Number of disk blocks read from the index in this table
idx_blks_hit	bigint	Number of buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

18.3.116 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays the I/O status information about all system catalog indexes in the namespace.

Table 18-188 PG_STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this index
idx_blks_hit	bigint	Number of buffer hits in this index

18.3.117 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays the I/O status information about all the system sequences in the namespace.

Table 18-189 PG_STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema this sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from this sequence
blks_hit	bigint	Number of buffer hits in this sequence

18.3.118 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES displays the I/O status information about all the system catalogs in the namespace.

Table 18-190 PG_STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from this table
heap_blks_hit	bigint	Number of buffer 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 buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

18.3.119 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays the I/O status information about all the user relationship table indexes in the namespace.

Table 18-191 PG_STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this index
idx_blks_hit	bigint	Number of buffer hits in this index

18.3.120 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES displays the I/O status information about all the user relation table sequences in the namespace.

Table 18-192 PG_STATIO_USER_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema 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

18.3.121 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays the I/O status information about all the user relation tables in the namespace.

Table 18-193 PG_STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table
idx_blks_read	bigint	Number of disk blocks read from the index in this table
idx_blks_hit	bigint	Number of buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

18.3.122 PG_THREAD_WAIT_STATUS

PG_THREAD_WAIT_STATUS allows you to test the block waiting status about the backend thread and auxiliary thread of the current instance.

Table 18-194 PG_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Current node name
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	Query ID. It is equivalent to debug_query_id .
tid	bigint	Thread ID of the current thread
lwtid	integer	Lightweight thread ID of the current thread
ptid	integer	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread

Name	Type	Description
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see Table 18-195 .
wait_event	text	If wait_status is acquire lock , acquire lwlock , or wait io , this column describes the lock, lightweight lock, and I/O information, respectively. If wait_status is not any of the three values, this column is empty.

The waiting statuses in the **wait_status** column are as follows:

Table 18-195 Waiting status list

Value	Description
none	Waiting for no event
acquire lock	Waiting for locking until the locking succeeds or times out
acquire lwlock	Waiting for a lightweight lock
wait io	Waiting for I/O completion
wait cmd	Waiting for network communication packet read to complete
wait pooler get conn	Waiting for pooler to obtain the connection
wait pooler abort conn	Waiting for pooler to terminate the connection
wait pooler clean conn	Waiting for pooler to clear connections
pooler create conn: [nodename], total N	Waiting for the pooler to set up a connection. The connection is being established with the node specified by <i>nodename</i> , and there are <i>N</i> connections waiting to be set up.
get conn	Obtaining the connection to other nodes
set cmd: [nodename]	Waiting for running the SET , RESET , TRANSACTION BLOCK LEVEL PARA SET , or SESSION LEVEL PARA SET statement on the connection. The statement is being executed on the node specified by <i>nodename</i> .
cancel query	Canceling the SQL statement that is being executed through the connection

Value	Description
stop query	Stopping the query that is being executed through the connection
wait node: [nodename](plevel), total N, [phase]	Waiting for receiving the data from a connected node. The thread is waiting for the data from the plevel thread of the node specified by <i>nodename</i>. The data of <i>N</i> connections is waiting to be returned. If <i>phase</i> is included, the possible phases are as follows: • begin: The transaction is being started. • commit: The transaction is being committed. • rollback: The transaction is being rolled back.
wait transaction sync: xid	Waiting for synchronizing the transaction specified by <i>xid</i>
wait wal sync	Waiting for the completion of wal log of synchronization from the specified LSN to the standby instance
wait data sync	Waiting for the completion of data page synchronization to the standby instance
wait data sync queue	Waiting for putting the data pages that are in the row storage or the CU in the column storage into the synchronization queue
flush data: [nodename](plevel), [phase]	Waiting for sending data to the plevel thread of the node specified by <i>nodename</i>. If <i>phase</i> is included, the possible phase is wait quota, indicating that the current communication flow is waiting for the quota value.
stream get conn: [nodename], total N	Waiting for connecting to the consumer object of the node specified by <i>nodename</i> when the stream flow is initialized. There are <i>N</i> consumers waiting to be connected.
wait producer ready: [nodename](plevel), total N	Waiting for each producer to be ready when the stream flow is initialized. The thread is waiting for the procedure of the plevel thread on the <i>nodename</i> node to be ready. There are <i>N</i> producers waiting to be ready.
synchronize quit	Waiting for the threads in the stream thread group to quit when the steam plan ends
nodegroup destroy	Waiting for destroying the stream node group when the steam plan ends

Value	Description
wait active statement	Waiting for job execution under resource and load control.
wait global queue	Waiting for job execution. The job is queuing in the global queue.
wait respool queue	Waiting for job execution. The job is queuing in the resource pool.
wait ccn queue	Waiting for job execution. The job is queuing on the central coordinator node (CCN).
gtm connect	Waiting for connecting to GTM.
gtm get gxid	Wait for obtaining xids from GTM.
gtm get snapshot	Wait for obtaining transaction snapshots from GTM.
gtm begin trans	Waiting for GTM to start a transaction.
gtm commit trans	Waiting for GTM to commit a transaction.
gtm rollback trans	Waiting for GTM to roll back a transaction.
gtm start prepare trans	Waiting for GTM to start the prepare phase of a two-phase transaction.
gtm prepare trans	Waiting for GTM to complete the prepare phase of a two-phase transaction.
gtm open sequence	Waiting for GTM to open a sequence.
gtm close sequence	Waiting for GTM to close a sequence.
gtm create sequence	Waiting for GTM to create a sequence.
gtm alter sequence	Waiting for GTM to modify a sequence.
gtm get sequence val	Waiting for obtaining the next value of a sequence from GTM.
gtm set sequence val	Waiting for GTM to set a sequence value.
gtm drop sequence	Waiting for GTM to delete a sequence.
gtm rename sequece	Waiting for GTM to rename a sequence.
analyze: [relname], [phase]	The thread is doing ANALYZE to the <i>relname</i> table. If <i>phase</i> is included, the possible phase is autovacuum , indicating that the database automatically enables the AutoVacuum thread to execute ANALYZE .

Value	Description
vacuum: [relname], [phase]	The thread is doing VACUUM to the <i>relname</i> table. If <i>phase</i> is included, the possible phase is autovacuum , indicating that the database automatically enables the AutoVacuum thread to execute VACUUM .
vacuum full: [relname]	The thread is doing VACUUM FULL to the <i>relname</i> table.
create index	An index is being created.
HashJoin - [build hash write file]	The HashJoin operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • build hash: The HashJoin operator is creating a hash table. • write file: The HashJoin operator is writing data to disks.
HashAgg - [build hash write file]	The HashAgg operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • build hash: The HashAgg operator is creating a hash table. • write file: The HashAgg operator is writing data to disks.
HashSetop - [build hash write file]	The HashSetop operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • build hash: The HashSetop operator is creating a hash table. • write file: The HashSetop operator is writing data to disks.
Sort Sort - write file	The Sort operator is being executed. write file indicates that the Sort operator is writing data to disks.
Material Material - write file	The Material operator is being executed. write file indicates that the Material operator is writing data to disks.
wait sync consumer next step	The consumer (receive end) synchronously waits for the next iteration.
wait sync producer next step	The producer (transmit end) synchronously waits for the next iteration.

If **wait_status** is **acquire lwlock**, **acquire lock**, or **wait io**, there is an event performing I/O operations or waiting for obtaining the corresponding lightweight lock or transaction lock.

The following table describes the corresponding wait events when **wait_status** is **acquire lwlock**. (If **wait_event** is **extension**, the lightweight lock is dynamically allocated and is not monitored.)

Table 18-196 List of wait events corresponding to lightweight locks

wait_event	Description
ShmemIndexLock	Used to protect the primary index table, a hash table, in shared memory
OidGenLock	Used to prevent different threads from generating the same OID
XidGenLock	Used to prevent two transactions from obtaining the same XID
ProcArrayLock	Used to prevent concurrent access to or concurrent modification on the ProcArray shared array
SInvalReadLock	Used to prevent concurrent execution with invalid message deletion
SInvalWriteLock	Used to prevent concurrent execution with invalid message write and deletion
WALInsertLock	Used to prevent concurrent execution with WAL insertion
WALWriteLock	Used to prevent concurrent write from a WAL buffer to a disk
ControlFileLock	Used to prevent concurrent read/write or concurrent write/write on the pg_control file
CheckpointLock	Used to prevent multi-checkpoint concurrent execution
CLogControlLock	Used to prevent concurrent access to or concurrent modification on the Clog control data structure
SubtransControlLock	Used to prevent concurrent access to or concurrent modification on the subtransaction control data structure
MultiXactGenLock	Used to allocate a unique MultiXact ID in serial mode
MultiXactOffsetControl-Lock	Used to prevent concurrent read/write or concurrent write/write on pg_multixact/offset
MultiXactMemberControl-Lock	Used to prevent concurrent read/write or concurrent write/write on pg_multixact/members

wait_event	Description
RelCacheInitLock	Used to add a lock before any operations are performed on the init file when messages are invalid
CheckpointInterCommLock	Used to send file flush requests to a checkpointer. The request structure needs to be inserted to a request queue in serial mode.
TwoPhaseStateLock	Used to prevent concurrent access to or modification on two-phase information sharing arrays
TablespaceCreateLock	Used to check whether a tablespace already exists
BtreeVacuumLock	Used to prevent VACUUM from clearing pages that are being used by B-tree indexes
AutovacuumLock	Used to access the autovacuum worker array in serial mode
AutovacuumScheduleLock	Used to distribute tables requiring VACUUM in serial mode
AutoanalyzeLock	Used to obtain and release resources related to a task that allows for autoanalyze execution
SyncScanLock	Used to determine the start position of a relfilenode during heap scanning
NodeTableLock	Used to protect a shared structure that stores CN and DN information
PoolerLock	Used to prevent two threads from simultaneously obtaining the same connection from a connection pool
RelationMappingLock	Used to wait for the mapping file between system catalogs and storage locations to be updated
AsyncCtlLock	Used to prevent concurrent access to or concurrent modification on the sharing notification status
AsyncQueueLock	Used to prevent concurrent access to or concurrent modification on the sharing notification queue
SerializableXactHashLock	Used to prevent concurrent read/write or concurrent write/write on a sharing structure for serializable transactions
SerializableFinishedList-Lock	Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions
SerializablePredicateLock-ListLock	Used to protect a linked list of serializable transactions that have locks

wait_event	Description
OldSerXidLock	Used to protect a structure that records serializable transactions that have conflicts
FileStatLock	Used to protect a data structure that stores statistics file information
SyncRepLock	Used to protect Xlog synchronization information during primary-standby replication
DataSyncRepLock	Used to protect data page synchronization information during primary-standby replication
CStoreColspaceCacheLock	Used to add a lock when CU space is allocated for a column-store table
CStoreCUCacheSweep-Lock	Used to add a lock when CU caches used by a column-store table are cyclically washed out
MetaCacheSweepLock	Used to add a lock when metadata is cyclically washed out
DfsConnectorCacheLock	Used to protect a global hash table where HDFS connection handles are cached
dummyServerInfoCache-Lock	Used to protect a global hash table where the information about computing Node Group connections is cached
ExtensionConnectorLibLock	Used to add a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios
SearchServerLibLock	Used to add a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios
DfsUserLoginLock	Used to protect a global linked table where HDFS user information is stored
DfsSpaceCacheLock	Used to ensure that the IDs of files to be imported to an HDFS table increase monotonically
LsnXlogChkFileLock	Used to serially update the Xlog flush points for primary and standby servers recorded in a specific structure
GTMHostInfoLock	Used to prevent concurrent access to or concurrent modification on GTM host information
ReplicationSlotAllocation-Lock	Used to add a lock when a primary server allocates stream replication slots during primary-standby replication
ReplicationSlotControl-Lock	Used to prevent concurrent update of replication slot status during primary-standby replication

wait_event	Description
ResourcePoolHashLock	Used to prevent concurrent access to or concurrent modification on a resource pool table, a hash table
WorkloadStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains SQL requests from the CN side
WorkloadIoStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains the I/O information of the current DN
WorkloadCGroupHash-Lock	Used to prevent concurrent access to or concurrent modification on a hash table that contains Cgroup information
OBSGetPathLock	Used to prevent concurrent read/write or concurrent write/write on an OBS path
WorkloadUserInfoLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains user information about load management
WorkloadRecordLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains requests received by CNs during adaptive memory management
WorkloadIOUtilLock	Used to protect a structure that records iostat and CPU load information
WorkloadNodeGroupLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains Node Group information in memory
JobShmemLock	Used to protect global variables in the shared memory that is periodically read during a scheduled task where MPP is compatible with Oracle
OBSRuntimeLock	Used to obtain environment variables, for example, GASSHOME
LLVMDumpIRLock	Used to export the assembly language for dynamically generating functions
LLVMParseIRLock	Used to compile and parse a finished IR function from the IR file at the start position of a query
RPNNumberLock	Used by a DN on a computing Node Group to count the number of threads for a task where plans are being executed
ClusterRPLock	Used to control concurrent access on cluster load data maintained in a CCN of the cluster

wait_event	Description
CriticalCacheBuildLock	Used to load caches from a shared or local cache initialization file
WaitCountHashLock	Used to protect a shared structure in user statement counting scenarios
BufMappingLock	Used to protect operations on a table mapped to shared buffer
LockMgrLock	It is used to protect a common lock structure.
PredicateLockMgrLock	Used to protect a lock structure that has serializable transactions
OperatorRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level
OperatorHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the operator level
SessionRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the query level
SessionHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the query level
CacheSlotMappingLock	Used to protect global CU cache information
BarrierLock	Used to ensure that only one thread is creating a barrier at a time

The following table describes the corresponding wait events when **wait_status** is **wait io**.

Table 18-197 List of wait events corresponding to I/Os

wait_event	Description
BufFileRead	Reads data from a temporary file to a specified buffer.
BufFileWrite	Writes the content of a specified buffer to a temporary file.
ControlFileRead	Reads the pg_control file, mainly during database startup, checkpoint execution, and primary/standby verification.

wait_event	Description
ControlFileSync	Flushes the pg_control file to a disk, mainly during database initialization.
ControlFileSyncUpdate	Flushes the pg_control file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileWrite	Writes to the pg_control file, mainly during database initialization.
ControlFileWriteUpdate	Updates the pg_control file, mainly during database startup, checkpoint execution, and primary/standby verification.
CopyFileRead	Reads a file during file copying.
CopyFileWrite	Writes a file during file copying.
DataFileExtend	Writes a file during file extension.
DataFileFlush	Flushes a table data file to a disk.
DataFileImmediateSync	Flushes a table data file to a disk immediately.
DataFilePrefetch	Reads a table data file asynchronously.
DataFileRead	Reads a table data file synchronously.
DataFileSync	Flushes table data file modifications to a disk.
DataFileTruncate	Truncates a table data file.
DataFileWrite	Writes a table data file.
LockFileAddToDataDir-Read	Reads the postmaster.pid file.
LockFileAddToDataDir-Sync	Flushes the postmaster.pid file to a disk.
LockFileAddToDataDir-Write	Writes the PID information into the postmaster.pid file.
LockFileCreateRead	Read the LockFile file %s.lock .
LockFileCreateSync	Flushes the LockFile file %s.lock to a disk.
LockFileCreateWRITE	Writes the PID information into the LockFile file %s.lock .
RelationMapRead	Reads the mapping file between system catalogs and storage locations.
RelationMapSync	Flushes the mapping file between system catalogs and storage locations to a disk.

wait_event	Description
RelationMapWrite	Writes the mapping file between system catalogs and storage locations.
ReplicationSlotRead	Reads a stream replication slot file during a restart.
ReplicationSlotRestore-Sync	Flushes a stream replication slot file to a disk during a restart.
ReplicationSlotSync	Flushes a temporary stream replication slot file to a disk during checkpoint execution.
ReplicationSlotWrite	Writes a temporary stream replication slot file during checkpoint execution.
SLRUFlushSync	Flushes the pg_clog , pg_subtrans , and pg_multixact files to a disk, mainly during checkpoint execution and database shutdown.
SLRURead	Reads the pg_clog , pg_subtrans , and pg_multixact files.
SLRUSync	Writes dirty pages into the pg_clog , pg_subtrans , and pg_multixact files, and flushes the files to a disk, mainly during checkpoint execution and database shutdown.
SLRUWrite	Writes the pg_clog , pg_subtrans , and pg_multixact files.
TimelineHistoryRead	Reads the timeline history file during database startup.
TimelineHistorySync	Flushes the timeline history file to a disk during database startup.
TimelineHistoryWrite	Writes to the timeline history file during database startup.
TwophaseFileRead	Reads the pg_twophase file, mainly during two-phase transaction submission and restoration.
TwophaseFileSync	Flushes the pg_twophase file to a disk, mainly during two-phase transaction submission and restoration.
TwophaseFileWrite	Writes the pg_twophase file, mainly during two-phase transaction submission and restoration.
WALBootstrapSync	Flushes an initialized WAL file to a disk during database initialization.
WALBootstrapWrite	Writes an initialized WAL file during database initialization.

wait_event	Description
WALCopyRead	Read operation generated when an existing WAL file is read for replication after archiving and restoration.
WALCopySync	Flushes a replicated WAL file to a disk after archiving and restoration.
WALCopyWrite	Write operation generated when an existing WAL file is read for replication after archiving and restoration.
WALInitSync	Flushes a newly initialized WAL file to a disk during log reclaiming or writing.
WALInitWrite	Initializes a newly created WAL file to 0 during log reclaiming or writing.
WALRead	Reads data from Xlogs during redo operations on two-phase files.
WALSyncMethodAssign	Flushes all open WAL files to a disk.
WALWrite	Writes a WAL file.

The following table describes the corresponding wait events when **wait_status** is **acquire lock**.

Table 18-198 List of wait events corresponding to transaction locks

wait_event	Description
relation	Adds a lock to a table.
extend	Adds a lock to a table being scaled out.
partition	Adds a lock to a partitioned table.
partition_seq	Adds a lock to a partition of a partitioned table.
page	Adds a lock to a table page.
tuple	Adds a lock to a tuple on a page.
transactionid	Adds a lock to a transaction ID.
virtualxid	Adds a lock to a virtual transaction ID.
object	Adds a lock to an object.
cstore_freespace	Adds a lock to idle column-store space.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.

18.3.123 PG_TABLES

PG_TABLES displays access to each table in the database.

Table 18-199 PG_TABLES columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE .nspname	Name of the schema that contains the table
tablename	name	PG_CLASS .relname	Name of the table
tableowner	name	pg_get_userbyid(PG_CLASS .relowner)	Owner of the table
tablespace	name	PG_TABLESPACE .spcname	Tablespace that contains the table. The default value is null
hasindexes	Boolean	PG_CLASS .relhasindex	Whether the table has (or recently had) an index. If it does, its value is true . Otherwise, its value is false .
hasrules	Boolean	PG_CLASS .relhasrules	Whether the table has rules. If it does, its value is true . Otherwise, its value is false .
hasindexes	Boolean	PG_CLASS .RELHASTRIGGERS	Whether the table has triggers. If it does, its value is true . Otherwise, its value is false .
tablecreator	name	pg_get_userbyid(PG_OBJECT .creator)	Table creator. If the creator has been deleted, no value is returned.
created	timestamp with time zone	PG_OBJECT .ctime	Time when the table was created.
last_ddl_time	timestamp with time zone	PG_OBJECT .mtime	Last time when the cluster was modified.

18.3.124 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays all time zone abbreviations that can be recognized by the input routines.

Table 18-200 PG_TIMEZONE_ABBREVS columns

Name	Type	Description
abbrev	text	Time zone abbreviation
utc_offset	interval	Offset from UTC
is_dst	Boolean	Whether the abbreviation indicates a daylight saving time (DST) zone. If it does, its value is true . Otherwise, its value is false .

18.3.125 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES displays all time zone names that can be recognized by **SET TIMEZONE**, along with their associated abbreviations, UTC offsets, and daylight saving time statuses.

Table 18-201 PG_TIMEZONE_NAMES columns

Name	Type	Description
name	text	Name of the time zone
abbrev	text	Time zone name abbreviation
utc_offset	interval	Offset from UTC
is_dst	Boolean	Whether DST is used. If it is, its value is true . Otherwise, its value is false .

18.3.126 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays the memory usage of a certain node in the database.

Table 18-202 PG_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name
memorytype	text	Memory name
memorybytes	integer	Size of the used memory in the unit of MB

18.3.127 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO displays the resource usage of all users. Only administrators can query this view. This view is valid only if **use_workload_manager** is set to **on**.

Table 18-203 PG_TOTAL_USER_RESOURCE_INFO columns

Name	Type	Description
username	name	Username
used_memory	integer	Used memory (unit: MB)
total_memory	integer	Available memory (unit: MB). 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. Only the CPU usage of complex jobs is collected, and the value is the CPU usage of the related Cgroup.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with a user on the node
used_space	bigint	Used permanent table storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB). -1 indicates that the storage space is not limited.
used_temp_space	bigint	Used temporary table storage space (unit: KB)
total_temp_space	bigint	Available temporary table storage space (unit: KB). -1 indicates that the storage space is not limited.
used_spill_space	bigint	Size of the used operator flushing space, in KB
total_spill_space	bigint	Size of the available operator flushing space, in KB. The value -1 indicates that the operator flushing space is not limited.
read_kbytes	bigint	CN: total number of bytes read by a user's complex jobs on all DN's in the last 5 seconds. The unit is KB. DN: total number of bytes read by a user's complex jobs from the instance startup time to the current time. The unit is KB.

Name	Type	Description
write_kbytes	bigint	CN: total number of bytes written by a user's complex jobs on all DN's in the last 5 seconds. The unit is KB. DN: total number of bytes written by a user's complex jobs from the instance startup time to the current time. The unit is KB.
read_counts	bigint	CN: total number of read times of a user's complex jobs on all DN's in the last 5 seconds. Unit: count. DN: total number of read times of a user's complex jobs from the instance startup time to the current time. Unit: count.
write_counts	bigint	CN: total number of write times of a user's complex jobs on all DN's in the last 5 seconds. Unit: count. DN: total number of write times of a user's complex jobs from the instance startup time to the current time. Unit: count.
read_speed	double precision	CN: average read rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s) DN: indicates the average read rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s)
write_speed	double precision	CN: average write rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s) DN: average write rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s)

18.3.128 PG_USER

PG_USER displays information about users who can access the database.

Table 18-204 PG_USER columns

Name	Type	Description
username	name	User name
usesysid	oid	ID of this user
usecreatedb	Boolean	Whether the user has the permission to create databases

Name	Type	Description
usesuper	Boolean	whether the user is the initial system administrator with the highest rights.
usecatupd	Boolean	whether the user can directly update system tables. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users.
userepl	Boolean	Whether the user has the permission to duplicate data streams
passwd	text	Encrypted user password. The value is displayed as *****.
valbegin	timestamp with time zone	Account validity start time; null if no start time
valuntil	timestamp with time zone	Password expiry time; null if no expiration
respool	name	Resource pool where the user is in
parentid	oid	Parent user OID
spacelimit	text	The storage space of the permanent table.
tempspacelimit	text	The storage space of the temporary table.
spillspacelimit	text	The operator disk flushing space.
useconfig	text[]	Session defaults for run-time configuration variables
nodegroup	name	Name of the logical cluster associated with the user. If no logical cluster is associated, this column is left blank.

18.3.129 PG_USER_MAPPINGS

PG_USER_MAPPINGS displays information about user mappings.

This is essentially a publicly readable view of **PG_USER_MAPPING** that leaves out the options column if the user has no rights to use it.

Table 18-205 PG_USER_MAPPINGS columns

Name	Type	Reference	Description
umid	oid	PG_USER_MAPPING .oid	OID of the user mapping
srvid	oid	PG_FOREIGN_SERVER .oid	OID of the foreign server that contains this mapping
srvname	name	PG_FOREIGN_SERVER .srvname	Name of the foreign server
umuser	oid	PG_AUTHID .oid	OID of the local role being mapped, 0 if the user mapping is public
username	name	-	Name of the local user to be mapped
umoptions	text[]	-	User mapping specific options. If the current user is the owner of the foreign server, its value is keyword=value strings. Otherwise, its value is null.

18.3.130 PG_VIEWS

PG_VIEWS displays basic information about each view in the database.

Table 18-206 PG_VIEWS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE .nspname	Name of the schema that contains the view
viewname	name	PG_CLASS .relname	View name
viewowner	name	PG_AUTHID .rolname	Owner of the view
definition	text	-	Definition of the view

18.3.131 PG_WLM_STATISTICS

PG_WLM_STATISTICS displays information about workload management after the task is complete or the exception has been handled.

Table 18-207 PG_WLM_STATISTICS columns

Name	Type	Description
statement	text	Statement executed for exception handling
block_time	bigint	Block time before the statement is executed
elapsed_time	bigint	Elapsed time when the statement is executed
total_cpu_time	bigint	Total time used by the CPU on the DN when the statement is executed for exception handling
qualification_time	bigint	Period when the statement checks the inclination ratio
cpu_skew_percent	integer	CPU usage skew on the DN when the statement is executed for exception handling
control_group	text	Cgroup used when the statement is executed for exception handling
status	text	Statement status after it is executed for exception handling ● pending: statement waiting to be executed ● running: statement being executed ● finished: execution normally finished ● aborted: execution unexpectedly terminated
action	text	Actions when statement are executed for exception handling ● abort: terminating the operation ● adjust: executing the Cgroup adjustment operations. Currently, you can only perform the demotion operation. ● finish: operation normally finished

18.3.132 PGXC_COMM_CLIENT_INFO

PGXC_COMM_CLIENT_INFO stores the client connection information of all nodes. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 18-208 PGXC_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	Client application name

Name	Type	Description
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	It 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	If the connection is a logical connection, sid is displayed. If -1 is displayed, the current connection is a physical connection.

18.3.133 PGXC_COMM_DELAY

PGXC_COMM_STATUS displays the communication library delay status for all the DNs.

Table 18-209 PGXC_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
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. Its unit is 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. Its unit is microsecond.
max_delay	integer	Maximum delay of the current physical connection within 1 minute. Its unit is microsecond.

18.3.134 PGXC_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of the communication libraries for all the DNs.

Table 18-210 PGXC_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	Current status of the stream
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. The unit is byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average receiving rate of the stream. The unit is byte/s.
quota	bigint	Current communication quota value of the stream. The unit is byte.
buff_usize	bigint	Current size of the data cache of the stream. The unit is byte.

18.3.135 PGXC_COMM_SEND_STREAM

PGXC_COMM_SEND_STREAM displays the sending stream status of the communication libraries for all the DNs.

Table 18-211 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	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	Current status of the stream
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. The unit is Byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average sending rate of the stream. The unit is Byte/s.
quota	bigint	Current communication quota value of the stream. The unit is Byte.
wait_quota	bigint	Extra time generated when the stream waits the quota value. The unit is ms.

18.3.136 PGXC_COMM_STATUS

PGXC_COMM_STATUS displays the communication library status for all the DNs.

Table 18-212 PGXC_COMM_STATUS columns

Name	Type	Description
node_name	text	Node name

Name	Type	Description
rxpck/s	integer	Receiving rate of the communication library on a node. The unit is byte/s.
txpck/s	integer	Sending rate of the communication library on a node. The unit is byte/s.
rxkB/s	bigint	Receiving rate of the communication library on a node. The unit is KB/s.
txkB/s	bigint	Sending rate of the communication library on a node. The unit is KB/s.
buffer	bigint	Size of the buffer of the Cmailbox.
memKB(libcomm)	bigint	Communication memory size of the libcomm process, in bytes.
memKB(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.

18.3.137 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 it.

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.

Table 18-213 PGXC_GET_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
relname	name	Table name
schemaname	name	Schema name of the table

Name	Type	Description
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
page_dirty_rate	numeric(5,2))	Dirty page rate (%) of a table

18.3.138 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS displays the data skew on tables in the current database.

Table 18-214 PGXC_GET_TABLE_SKEWNESS columns

Name	Type	Description
schemaname	name	Schema name of a table
tablename	name	Name of a table
totalsize	numeric	Total size of a table, in bytes
avgsz	numeric(1000,0)	Average table size (total table size divided by the number of DNs), which is the ideal size of tables distributed on each DN
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.)

18.3.139 PGXC_GTM_SNAPSHOT_STATUS

PGXC_GTM_SNAPSHOT_STATUS displays transaction information on the current GTM.

Table 18-215 PGXC_GTM_SNAPSHOT_STATUS columns

Name	Type	Description
xmin	xid	Minimum ID of the running transactions
xmax	xid	ID of the transaction next to the executed transaction with the maximum ID
csn	integer	Sequence number of the transaction to be committed
oldestxmin	xid	Minimum ID of the executed transactions
xcnt	integer	Number of the running transactions
running_xids	text	IDs of the running transactions

18.3.140 PGXC_INSTR_UNIQUE_SQL

PGXC_INSTR_UNIQUE_SQL displays the complete Unique SQL statistics of all CN nodes in the cluster.

Only the system administrator can access this view. For details about the field, see [GS_INSTR_UNIQUE_SQL](#).

18.3.141 PGXC_NODE_ENV

PGXC_NODE_ENV displays the environmental variables information about all nodes in a cluster.

Table 18-216 PGXC_NODE_ENV columns

Name	Type	Description
node_name	text	Names of all nodes in the cluster
host	text	Host names of all nodes in the cluster
process	integer	Process IDs of all nodes in the cluster
port	integer	Port numbers of all nodes in the cluster
installpath	text	Installation directory of all nodes in the cluster
datapath	text	Data directory of all nodes in the cluster
log_directory	text	Log directory of all nodes in the cluster

18.3.142 PGXC_OS_THREADS

PGXC_OS_THREADS displays thread status information under all normal nodes in the current cluster.

Table 18-217 PGXC_OS_THREADS columns

Name	Type	Description
node_name	text	All normal node names in the cluster
pid	bigint	IDs of running threads among all normal node processes in the current cluster
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

18.3.143 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays the two-phase transactions in the **prepared** phase.

Table 18-218 PGXC_PREPARED_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Two-phase transactions in prepared phase

18.3.144 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**.

Table 18-219 PGXC_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 or 0 : starting)

Name	Type	Description
node	text	Node name
xmin	xid	Minimum transaction ID xmin on the node
vacuum	Boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restart
prepare_xid	xid	Transaction ID in 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

18.3.145 PGXC_STAT_ACTIVITY

PGXC_STAT_ACTIVITY displays the query information about the current user on all the CNs in the current cluster. Only the system administrator has the permission to query this view.

Table 18-220 PGXC_STAT_ACTIVITY columns

Name	Type	Description
coorname	text	Name of the CN in the current cluster
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	ID of the backend thread
usesysid	oid	OID of the user logging in to the backend
username	name	Name of the user logging in to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend null 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.

Name	Type	Description
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null when IP connections are used and log_hostname is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server.
xact_start	timestamp with time zone	Time when the current transaction was started, or NULL if no transaction is active. If the current query is the first of its transaction, this column is equal to the query_start column.
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 for the last status change
waiting	Boolean	Whether the backend is currently waiting on a lock. If it is, its value is true .
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	Current overall state 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 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;</pre> <table><thead><tr><th>datname</th><th>username</th><th>usesysid</th><th>state</th><th>pid</th></tr></thead><tbody><tr><td>postgres</td><td>omm</td><td>10</td><td></td><td>139968752121616</td></tr><tr><td>postgres</td><td>omm</td><td>10</td><td></td><td>139968903116560</td></tr><tr><td>db_tpcds</td><td>judy</td><td>16398</td><td>active</td><td>139968391403280</td></tr><tr><td>postgres</td><td>omm</td><td>10</td><td></td><td>139968643069712</td></tr><tr><td>postgres</td><td>omm</td><td>10</td><td></td><td>139968680818448</td></tr><tr><td>postgres</td><td>joe</td><td>16390</td><td></td><td>139968563377936</td></tr></tbody></table> (6 rows)	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
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																																	
resource_pool	name	Resource pool used by the user																																			
query_id	bigint	ID of a query																																			
query	text	Recent query of this backend. If state is active , this column shows the executing query. In all other states, it shows the last query that was executed.																																			
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database (for details, see connection_info)																																			

18.3.146 PGXC_STAT_BAD_BLOCK

PGXC_STAT_BAD_BLOCK displays statistics about page or CU verification failures after all nodes in a cluster are started.

Table 18-221 PGXC_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
forknum	integer	File type
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first occurrence
last_time	timestamp with time zone	Time of the latest occurrence

18.3.147 PGXC_SQL_COUNT

PGXC_SQL_COUNT displays the node-level and user-level statistical results for the SQL statements of **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** and DDL, DML, and DCL statements in real time, identifies query types with heavy load, and measures the capability of a cluster or a node to perform a specific type of query. For example, you can calculate QPS and collect statistics on the response time based on the quantities of the several types of SQL statements at certain time points. For example, if the **SELECT** count of **USER1** at **T1** is **X1**, and that at **T2** is **X2**, the QPS of **SELECT** for **USER1** is $(X2 - X1) / (T2 - T1)$. In this way, the system can draw cluster-user-level QPS curve graphs and determine cluster throughput, monitoring 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.

Columns in the **PGXC_SQL_COUNT** view are the same as those in the **GS_SQL_COUNT** view. For details, see [Table 18-123](#).

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 increment by 1. If the pushdown is not allowed, the DN will receive an **UPDATE** or **INSERT** statement. In this case, the **update_count** or **insert_count** column will increment by 1.

18.3.148 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** view and **PG_THREAD_WAIT_STATUS** view are the same, because the essence of the **PGXC_THREAD_WAIT_STATUS** view is the query summary result of the **PG_THREAD_WAIT_STATUS** view on each node in the cluster.

Table 18-222 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
lwtid	integer	Lightweight thread ID of the current thread
ptid	integer	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see Table 18-195 .
wait_event	text	If wait_status is acquire lock , acquire lwlock , or wait io , this column describes the lock, lightweight lock, and I/O information, respectively. If wait_status is not any of the three values, this column is empty.

Example:

Assume you run a statement on coordinator1, and no response is returned after a long period of time. In this case, establish another connection to coordinator1 to check the thread status on it.

```
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)
```

Furthermore, you can view the statement working status on each node in the entire cluster. In the following example, no DNs have threads blocked, and there is a huge amount of data to be read, causing slow execution.

```
select * from pgxc_thread_wait_status where query_id=20971544;
 node_name | db_name | thread_name | query_id | tid | lwtid | ptid | tlevel | smpid |
wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | postgres | coordinator1 | 20971544 | 139902867994384 | 22735 | | 0 | 0 | wait
node: datanode3 |
datanode1 | postgres | coordinator1 | 20971544 | 139902838634256 | 22970 | 22735 | 5 | 0 |
synchronize quit |
datanode1 | postgres | coordinator1 | 20971544 | 139902607947536 | 22972 | 22735 | 5 | 1 |
synchronize quit |
datanode2 | postgres | coordinator1 | 20971544 | 140632156796688 | 22736 | | 0 | 0 | wait
node: datanode3 |
datanode2 | postgres | coordinator1 | 20971544 | 140632030967568 | 22974 | 22736 | 5 | 0 |
synchronize quit |
datanode2 | postgres | coordinator1 | 20971544 | 140632081299216 | 22975 | 22736 | 5 | 1 |
synchronize quit |
datanode3 | postgres | coordinator1 | 20971544 | 140323627988752 | 22737 | | 0 | 0 | wait
node: datanode3 |
datanode3 | postgres | coordinator1 | 20971544 | 140323523131152 | 22976 | 22737 | 5 | 0 | net
flush data |
datanode3 | postgres | coordinator1 | 20971544 | 140323548296976 | 22978 | 22737 | 5 | 1 | net
flush data |
datanode4 | postgres | coordinator1 | 20971544 | 140103024375568 | 22738 | | 0 | 0 | wait
node: datanode3 |
datanode4 | postgres | coordinator1 | 20971544 | 140102919517968 | 22979 | 22738 | 5 | 0 |
synchronize quit |
datanode4 | postgres | coordinator1 | 20971544 | 140102969849616 | 22980 | 22738 | 5 | 1 |
synchronize quit |
coordinator1 | postgres | gsql | 20971544 | 140274089064208 | 22579 | | 0 | 0 | wait node:
datanode4 |
(13 rows)
```

18.3.149 PGXC_TOTAL_MEMORY_DETAIL

PGXC_TOTAL_MEMORY_DETAIL displays the memory usage in the cluster.

Table 18-223 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(DWS) cluster instance • process_used_memory: memory occupied by a GaussDB(DWS) process • max_dynamic_memory: maximum dynamic memory • dynamic_used_memory: used dynamic memory • dynamic_peak_memory: dynamic peak value of the 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_cstore_memory: maximum memory allowed by column storage • cstore_used_memory: memory used in column storage • max_sctpcomm_memory: maximum memory size allowed for the communication library • sctpcomm_used_memory: memory used by communication library • sctpcomm_peak_memory: memory peak of the communication library • other_used_memory: other used memory
memorybytes	integer	Size of the used memory in the unit of MB

18.3.150 PGXC_VARIABLE_INFO

PGXC_VARIABLE_INFO displays information about transaction IDs and OIDs of all nodes in a cluster.

Table 18-224 PGXC_VARIABLE_INFO columns

Name	Type	Description
node_name	text	Node name
nextOid	oid	OID generated next time for a node
nextXid	xid	Transaction ID generated next time for a node
oldestXid	xid	Oldest transaction ID for a node
xidVacLimit	xid	Critical point that triggers forcible autovacuum
oldestXidDB	oid	OID of the database that has the minimum datafrozenxid on a node
lastExtendCSNL ogpage	integer	Number of the last extended cslog page
startExtendCSN Logpage	integer	Number of the page from which the cslog extending starts
nextCommitSeq No	integer	CSN generated next time for a node
latestCompleted Xid	xid	Latest transaction ID on a node after the transaction commission or rollback
startupMaxXid	xid	Last transaction ID before a node is powered off

18.3.151 PGXC_WLM_OPERATOR_HISTORY

PGXC_WLM_OPERATOR_HISTORY displays the operator information of completed jobs executed on all CNs. This view is used by Database Manager to query data from a database. Data in the database is cleared every 3 minutes.

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-3](#).

18.3.152 PGXC_WLM_OPERATOR_INFO

PGXC_WLM_OPERATOR_INFO displays the operator information of completed jobs executed on CNs. The data in this view is obtained from [GS_WLM_OPERATOR_INFO](#).

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-3](#).

18.3.153 PGXC_WLM_OPERATOR_STATISTICS

PGXC_WLM_OPERATOR_STATISTICS displays the operator information of jobs being executed on CNs.

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-125](#).

18.3.154 PGXC_WLM_SESSION_INFO

PGXC_WLM_SESSION_INFO displays load management information for completed jobs executed on all CNs. The data in this view is obtained from [GS_WLM_SESSION_INFO](#).

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-126](#).

18.3.155 PGXC_WLM_SESSION_HISTORY

PGXC_WLM_SESSION_HISTORY displays load management information for completed jobs executed on all CNs. This view is used by Data Manager to query data from a database. Data in the database is cleared every 3 minutes. For details, see [GS_WLM_SESSION_HISTORY](#).

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-126](#).

18.3.156 PGXC_WLM_SESSION_STATISTICS

PGXC_WLM_SESSION_STATISTICS displays load management information about jobs that are being executed on CNs.

This view is accessible only to users with system administrators rights. For details about columns in the view, see [Table 18-127](#).

18.3.157 PGXC_WLM_WORKLOAD_RECORDS

PGXC_WLM_WORKLOAD_RECORDS displays the status of job executed by the current user on CNs. It is accessible only to users with system administrator rights. This view is available only when **enable_dynamic_workload** is set to **on**.

Table 18-225 PGXC_WLM_WORKLOAD_RECORDS columns

Name	Type	Description
node_name	text	Name of the CN where the job is executed
thread_id	bigint	ID of the backend thread
processid	integer	lwpid of a thread
timestamp	bigint	Time when a statement starts to be executed
username	name	Name of the user logging in to the backend
memory	integer	Memory required by a statement
active_points	integer	Number of resources consumed by a statement in a resource pool

Name	Type	Description
max_points	integer	Maximum number of resources in a resource pool
priority	integer	Priority of a job
resource_pool	text	Resource pool to which a job belongs
status	text	Job execution status. Its value can be: pending running finished aborted unknown
control_group	text	Cgroups used by a job
enqueue	text	Queue that a job is in. Its value can be: GLOBAL : global queue RESPOOL : resource pool queue ACTIVE : not in a queue
query	text	Statement that is being executed

18.3.158 GS_WORKLOAD_SQL_COUNT

GS_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on the current node, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 18-226 GS_WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload Cgroup 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

Name	Type	Description
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements

18.3.159 GS_WORKLOAD_SQL_ELAPSE_TIME

GS_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on the current node, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 18-227 GS_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload Cgroup name
total_select_elapse	bigint	Total response time of SELECT statements
max_select_elapse	bigint	Maximum response time of SELECT statements
min_select_elapse	bigint	Minimum response time of SELECT statements
avg_select_elapse	bigint	Average response time of SELECT statements
total_update_elapse	bigint	Total response time of UPDATE statements
max_update_elapse	bigint	Maximum response time of UPDATE statements
min_update_elapse	bigint	Minimum response time of UPDATE statements
avg_update_elapse	bigint	Average response time of UPDATE statements
total_insert_elapse	bigint	Total response time of INSERT statements
max_insert_elapse	bigint	Maximum response time of INSERT statements

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of INSERT statements
avg_insert_elapse	bigint	Average response time of INSERT statements
total_delete_elapse	bigint	Total response time of DELETE statements
max_delete_elapse	bigint	Maximum response time of DELETE statements
min_delete_elapse	bigint	Minimum response time of DELETE statements
avg_delete_elapse	bigint	Average response time of DELETE statements

18.3.160 PLAN_TABLE

PLAN_TABLE displays the plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level life cycle. After the session exits, the data will be deleted. Data is isolated between sessions and between users.

Table 18-228 PLAN_TABLE columns

Name	Type	Description
statement_id	varchar2(30)	Query tag specified by a user
plan_id	bigint	ID of a plan to be queried
id	int	ID of each operator in a generated plan
operation	varchar2(30)	Operation description of an operator in a plan
options	varchar2(255)	Operation parameters
object_name	name	Name of an operated object. It is defined by users, not the object alias used in the query.
object_type	varchar2(30)	Object type
object_owner	name	User-defined schema to which an object belongs
projection	varchar2(4000)	Returned column information

 NOTE

- A valid **object_type** value consists of a relkind type defined in **PG_CLASS** (**TABLE** ordinary table, **INDEX**, **SEQUENCE**, **VIEW**, **FOREIGN TABLE**, **COMPOSITE TYPE**, or **TOASTVALUE TOAST** table) 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.

18.3.161 PLAN_TABLE_DATA

PLAN_TABLE_DATA displays the plan information collected by **EXPLAIN PLAN**. Different from the **PLAN_TABLE** view, the system catalog **PLAN_TABLE_DATA** stores the plan information collected by all sessions and users.

Table 18-229 PLAN_TABLE columns

Name	Type	Description
session_id	text	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by NOT NULL .
user_id	oid	User who inserts the data. Values are constrained by NOT NULL .
statement_id	varchar2(30)	Query tag specified by a user
plan_id	bigint	ID of a plan to be queried
id	int	Node ID in a plan
operation	varchar2(30)	Operation description
options	varchar2(255)	Operation parameters
object_name	name	Name of an operated object. It is defined by users.
object_type	varchar2(30)	Object type
object_owner	name	User-defined schema to which an object belongs
projection	varchar2(4000)	Returned column information

 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 only 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 from **PLAN_TABLE_DATA**.
- 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 **PLAN_TABLE_DATA**, 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.

18.3.162 PV_FILE_STAT

By collecting statistics about the data file I/Os, **PV_FILE_STAT** displays the I/O performance of the data to detect the performance problems, such as abnormal I/O operations.

Table 18-230 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
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 files. The unit is microsecond.
writetim	bigint	Total duration of writing files. The unit is microsecond.
avgiotim	bigint	Average duration of reading and writing files. The unit is microsecond.
lstiotim	bigint	Duration of the last file reading. The unit is microsecond.
miniotim	bigint	Minimum duration of reading and writing files. The unit is microsecond.

Name	Type	Description
maxiowtm	bigint	Maximum duration of reading and writing files. The unit is microsecond.

18.3.163 PV_INSTANCE_TIME

PV_INSTANCE_TIME collects statistics on the running time of processes and the time consumed in each execution phase, in microseconds.

Table 18-231 PV_INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID
stat_name	text	Running time type name
value	bigint	Running time value

18.3.164 PV_OS_RUN_INFO

PV_OS_RUN_INFO displays the running status of the current operating system.

Table 18-232 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

18.3.165 PV_SESSION_MEMORY

PV_SESSION_MEMORY displays statistics about memory usage at the session level in the unit of MB, including all the memory allocated to Postgres and Stream threads on DN for jobs currently executed by users.

Table 18-233 PV_SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed task before the task enters the executor, in MB
used_mem	integer	Memory allocated to the currently executed task, in MB
peak_mem	integer	Peak memory allocated to the currently executed task, in MB

18.3.166 PV_SESSION_MEMORY_DETAIL

PV_SESSION_MEMORY_DETAIL displays statistics about thread memory usage by memory context.

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 */tmp/dumpmem* directory. *threadid* can be obtained from the following table.

Table 18-234 PV_SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	Thread start time+thread ID (string: <i>timestamp.threadid</i>)
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes
freesize	bigint	Total size of released memory in the memory context, in bytes

Name	Type	Description
usedsize	bigint	Size of used memory in the memory context, in bytes. For TempSmallContextGroup, this parameter specifies the number of collected memory contexts.

18.3.167 PV_SESSION_STAT

PV_SESSION_STAT displays session state statistics based on session threads or the **AutoVacuum** thread.

Table 18-235 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

18.3.168 PV_SESSION_TIME

PV_SESSION_TIME displays statistics about the running time of session threads and time consumed in each execution phase, in microseconds.

Table 18-236 PV_SESSION_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time
stat_id	integer	Statistics ID
stat_name	text	Running time type name
value	bigint	Running time value

18.3.169 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

Table 18-237 PV_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name
memorytype	text	Memory type. Its value can be: • max_process_memory: memory occupied by a GaussDB(DWS) cluster instance • process_used_memory: memory occupied by a GaussDB(DWS) process • max_dynamic_memory: maximum dynamic memory • dynamic_used_memory: used dynamic memory • dynamic_peak_memory: dynamic peak value of the 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_cstore_memory: maximum memory allowed by column storage • cstore_used_memory: memory used in column storage • max_sctpcomm_memory: maximum memory size allowed for the communication library • sctpcomm_used_memory: memory used by communication library • sctpcomm_peak_memory: memory peak of the communication library • other_used_memory: other used memory
memorybytes	integer	Size of allocated memory-typed memory

18.3.170 REDACTION_COLUMNS

REDACTION_COLUMNS displays information about all redaction columns in the current database.

Table 18-238 REDACTION_COLUMNS columns

Name	Type	Description
object_owner	name	Owner of the object to be redacted.
object_name	name	Redacted object name
column_name	name	Redacted column name
function_type	integer	Redaction type
function_parameters	text	Parameters when the redaction type is partial
regexp_pattern	text	Pattern string when the redaction type is regexp
regexp_replace_string	text	Replacement string when the redaction type is regexp
regexp_position	integer	Specifies the start replacement position when the redaction type is regexp
regexp_occurrence	integer	The number of replacements when the redaction type is regexp
regexp_match_parameter	text	Regular control parameter when the redaction type is regexp
column_description	text	Description of the redacted column

18.3.171 REDACTION_POLICIES

REDACTION_POLICIES displays information about all redaction objects in the current database.

Table 18-239 REDACTION_POLICIES columns

Name	Type	Description
object_owner	name	Owner of the object to be redacted.
object_name	name	Redacted object name
policy_name	name	Name of the redact policy

Name	Type	Description
expression	text	Policy effective expression (for users)
enable	boolean	Policy status (enabled or disabled)
policy_description	text	Description of a policy

18.3.172 USER_COL_COMMENTS

USER_COL_COMMENTS displays the column comments of the table accessible to the current user.

Table 18-240 USER_COL_COMMENTS columns

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table name
owner	character varying(64)	Table owner
comments	text	Comments

18.3.173 USER_CONSTRAINTS

USER_CONSTRAINTS displays the table constraint information accessible to the current user.

Table 18-241 USER_CONSTRAINTS columns

Name	Type	Description
constraint_name	vcharacter 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 constraint-related table

Name	Type	Description
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

18.3.174 USER_CONS_COLUMNS

USER_CONSTRAINTS displays the information about constraint columns of the tables accessible to the current user.

Table 18-242 USER_CONS_COLUMNS columns

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

18.3.175 USER_INDEXES

USER_INDEXES displays index information in the current schema.

Table 18-243 USER_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_name	character varying(64)	Table name for the index
uniqueness	text	Whether the index is a unique index
generated	character varying(1)	Whether the index name is generated by the system

Name	Type	Description
partitioned	character(3)	Whether the index has the property of the partition table

18.3.176 USER_IND_COLUMNS

USER_IND_COLUMNS displays column information about all indexes accessible to the current user.

Table 18-244 USER_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 column in the index

18.3.177 USER_IND_EXPRESSIONS

USER_IND_EXPRESSIONS displays information about the function-based expression index accessible to the current user.

Table 18-245 USER_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
column_position	smallint	Position of column in the index

18.3.178 USER_IND_PARTITIONS

USER_IND_PARTITIONS displays information about index partitions accessible to the current user.

Table 18-246 USER_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which the index partition belongs
schema	character varying(64)	Schema of the partitioned table index to which the 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
index_partition_usable	Boolean	Whether the index partition is available
high_value	text	Upper limit of the partition corresponding to the index partition
def_tablespace_name	name	Name of the tablespace of the index partition

18.3.179 USER_JOBS

USER_JOBS displays all jobs owned by the user.

Table 18-247 USER_JOBS columns

Name	Type	Description
job	int4	Job ID
log_user	name not null	User name of the job creator
priv_user	name not null	User name of the job executor
dbname	name not null	Database in which the job is created

Name	Type	Description
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
this_date	timestamp without time zone	Start time of the ongoing job execution
this suc	text	Same as THIS_DATE
next_date	timestamp without time zone	Schedule time of the next job execution
next suc	text	Same as next_date
broken	text	Task status Y : the system does not try to execute the task. N : the system attempts to execute the task.
status	char	Status of the current job. The value range is 'r', 's', 'f', 'd'. The default value is 's'. The indications are as follows: • r: running • s: finished • f: failed • d: aborted
interval	text	Time expression used to calculate the next execution time. 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	Body of the PL/SQL blocks or anonymous block that the job executes

18.3.180 USER_OBJECTS

USER_OBJECTS displays all database objects accessible to the current user.

Table 18-248 USER_OBJECTS columns

Name	Type	Description
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object (TABLE , INDEX , SEQUENCE , or VIEW)
namespace	oid	Namespace that the object belongs to
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.181 USER_PART_INDEXES

USER_PART_INDEXES displays information about partitioned table indexes accessible to the current user.

Table 18-249 USER_PART_INDEXES columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index
schema	character varying(64)	Schema of the partitioned table index
index_name	character varying(64)	Name of the partitioned table index
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs
partitioning_type	text	Partition policy of the partitioned table

Name	Type	Description
partition_count	bigint	Number of index partitions of the partitioned table index
def_tablespace_name	name	Name of the tablespace of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table

18.3.182 USER_PART_TABLES

USER_PART_TABLES displays information about partitioned tables accessible to the current user.

Table 18-250 USER_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Name of the owner of the partitioned table
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Name of the partitioned table
partitioning_type	text	Partition policy of the partitioned table
partition_count	bigint	Number of partitions of the partitioned table
def_tablespace_name	name	Name of the tablespace of the partitioned table
partitioning_key_count	integer	Number of partition keys of the partitioned table

18.3.183 USER_PROCEDURES

USER_PROCEDURES displays information about all stored procedures and functions in the current schema.

Table 18-251 USER_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function

Name	Type	Description
object_name	character varying(64)	Name of the stored procedure or the function
argument_number	smallint	Number of the input parameters in the stored procedure

18.3.184 USER_SEQUENCES

USER_SEQUENCES displays sequence information in the current schema.

Table 18-252 USER_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of the sequence
sequence_name	character varying(64)	Name of the sequence

18.3.185 USER_SOURCE

USER_SOURCE displays information about stored procedures or functions in this mode, and provides the columns defined by the stored procedures or the functions.

Table 18-253 USER_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
name	character varying(64)	Name of the stored procedure or the function
text	text	Definition of the stored procedure or the function

18.3.186 USER_SYNONYMS

USER_SYNONYMS displays synonyms accessible to the current user.

Table 18-254 USER_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.
table_schema_name	text	Schema name of the associated object.
table_name	text	Name of the associated object.

18.3.187 USER_TAB_COLUMNS

USER_TAB_COLUMNS displays information about table columns accessible to the current user.

Table 18-255 USER_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
column_id	integer	Sequence number of the column when the table is created
data_length	integer	Length of the column in the unit of bytes
comments	text	Comments
avg_col_len	numeric	Average length of a column in the unit of bytes
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n .

Name	Type	Description
data_precision	integer	Precision of the data type. This parameter is valid for the numeric data type and NULL for other types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type and 0 for other types.
char_length	numeric	Column length in the unit of bytes which is valid only for the varchar, nvarchar2, bpchar, and char types.

18.3.188 USER_TAB_COMMENTS

USER_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

Table 18-256 USER_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of the table or the view
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

18.3.189 USER_TAB_PARTITIONS

USER_TAB_PARTITIONS displays all table partitions accessible to the current user. Each partition of a partitioned table accessible to the current user has a piece of record in **USER_TAB_PARTITIONS**.

Table 18-257 USER_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Name of the owner of the partitioned table
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Name of the partitioned table

Name	Type	Description
partition_name	character varying(64)	Name of the table partition
high_value	text	Upper boundary of the table partition
tablespace_name	name	Name of the tablespace of the table partition

18.3.190 USER_TABLES

USER_TABLES displays table information in the current schema.

Table 18-258 USER_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace where the table is located
status	character varying(8)	Whether the current record is valid
temporary	character(1)	Whether the table is a temporary table • Y: temporary table • N: not a temporary table
dropped	character varying	Whether the current record is deleted • YES: deleted • NO: not deleted
num_rows	numeric	Estimated number of rows in the table

18.3.191 USER_TRIGGERS

USER_TRIGGERS displays the information about triggers accessible to the current user.

Table 18-259 USER_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_name	character varying(64)	Name of the relationship table
table_owner	character varying(64)	Role name

18.3.192 USER_VIEWS

USER_VIEWS displays information about all views in the current schema.

Table 18-260 USER_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of the view
view_name	character varying(64)	View name

18.3.193 V\$SESSION

V\$SESSION displays all session information about the current session.

Table 18-261 V\$SESSION columns

Name	Type	Description
sid	bigint	OID of the background process of the current activity
serial#	integer	Sequence number of the active background process, which is 0 in GaussDB(DWS).
user#	oid	OID of the user that has logged in to the background process
username	name	Name of the user that has logged in to the background process

18.3.194 V\$SESSION_LONGOPS

V\$SESSION_LONGOPS displays the progress of ongoing operations.

Table 18-262 V\$SESSION_LONGOPS columns

Name	Type	Description
sid	bigint	OID of the running background process
serial#	integer	Sequence number of the running background process, which is 0 in GaussDB(DWS).
sofar	integer	Completed workload, which is empty in GaussDB(DWS).
totalwork	integer	Total workload, which is empty in GaussDB(DWS).

19 Information Schema

The information schema is a schema named `information_schema`. It automatically exists in all databases. The information schema consists of a set of views that contain information that defines objects in the current database. The owner of this schema is an initial database user, and the user naturally owns all the privileges of this schema, including the ability to delete it.

20 Tool Reference

While you use GaussDB(DWS), you need to install, upgrade, scale out, and uninstall clusters, and to manage cluster health. To facilitate cluster maintenance, GaussDB(DWS) provides a set of database management tools.

20.1 gs_dump

Context

gs_dump, provided by GaussDB(DWS), is used to export database information. Users can export a database or its objects (such as schemas, tables, and views). The database can be the default **postgres** database or a user-specified database.

When **gs_dump** is used to export data, other users still can access the database (readable or writable).

gs_dump can export complete, consistent data. For example, if **gs_dump** is started to export database A at T1, data of the database at that time point will be exported, and modifications on the database after that time point will not be exported.

gs_dump can export database information to a plain-text SQL script file or archive file.

- Plain-text SQL script: It contains the SQL statements required to restore the database. You can use **gsql** to execute the SQL script. With only a little modification, the SQL script can rebuild a database on other hosts or database products.
- Archive file: It contains data required to restore the database. It can be a tar-, directory-, or custom-format archive. For details, see [Table 20-1](#). The export result must be used with **gs_restore** to restore the database. The system allows users to select or even to sort the content to be imported.

Functions

gs_dump can create export files in four formats, which are specified by **-F** or **--format=**, as listed in [Table 20-1](#).

Table 20-1 Formats of exported files

Format	Value of -F	Description	Suggestion	Corresponding Import Tool
Plain-text	p	A plain-text script file containing SQL statements and commands. The commands can be executed on gsql , a command line terminal, to recreate database objects and load table data.	You are advised to use plain-text export files for small databases.	Before using gsql to restore database objects, you can use a text editor to edit the plain-text export file as needed.
Custom	c	A binary file that allows the restoration of all or selected database objects from an exported file.	You are advised to use custom-format archive files for medium or large database.	You can use gs_restore to import database objects from a custom-format archive.
Directory	d	A directory containing directory files and the data files of tables and BLOB objects.	-	
.tar	t	A tar-format archive that allows the restoration of all or selected database objects from an exported file. It cannot be further compressed and has an 8-GB limitation on the size of a single table.	-	

 NOTE

To reduce the size of an exported file, you can use the **gs_dump** tool to compress it to a plain-text file or custom-format file. By default, a plain-text file is not compressed when generated. When a custom-format archive is generated, a medium level of compression is applied by default. Archived exported files cannot be compressed using **gs_dump**.

Precautions

Do not modify an exported file or its content. Otherwise, restoration may fail.

To ensure the data consistency and integrity, **gs_dump** acquires a share lock on a table to be dumped. If another transaction has acquired a share lock on the table,

gs_dump waits until this lock is released and then locks the table for dumping. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.

Syntax

```
gs_dump [OPTION]... [DBNAME]
```

NOTE

DBNAME does not follow a short or long option. It specifies the database to connect to.

For example:

Specify *DBNAME* without a **-d** option preceding it.

```
gs_dump -p port_number postgres -f dump1.sql
```

or

```
export PGDATABASE=postgres
```

```
gs_dump -p port_number -f dump1.sql
```

Environment variable: *PGDATABASE*

Parameter Description

Common parameters:

- **-f, --file=FILENAME**
Sends the output to the specified file or directory. If this parameter is omitted, the standard output is generated. If the output format is **(-F c/-F d/-F t)**, the **-f** parameter must be specified. If the value of the **-f** parameter contains a directory, the directory has the read and write permissions to the current user.
- **-F, --format=c|d|t|p**
Selects the exported file format. Its format can be:
 - **p|plain**: Generates a text SQL script file. This is the default value.
 - **c|custom**: Outputs a custom-format archive as a directory to be used as the input of **gs_restore**. This is the most flexible output format in which users can manually select it and reorder the archived items during the restore process. An archive in this format is compressed by default.
 - **d|directory**: A directory containing directory files and the data files of tables and BLOB objects.
 - **t|tar**: Outputs a tar format as the archive form that is suitable for the input of **gs_restore**. The .tar format is compatible with the directory format. Extracting a .tar archive generates a valid directory-format archive. However, the .tar archive cannot be further compressed and has an 8-GB limitation on the size of a single table. The order of table data items cannot be changed during restoration.
A .tar archive can be used as input of **gsqsl**.
- **-v, --verbose**
Specifies the verbose mode. If it is specified, **gs_dump** writes detailed object comments and the number of startups/stops to the dump file, and progress messages to standard error.
- **-V, --version**
Prints the *gs_dump* version and exits.

- **-Z, --compress=0-9**
Specifies the used compression level.
Value range: 0 to 9
 - **0** indicates no compression.
 - **1** indicates that the compression ratio is the lowest and processing speed the fastest.
 - **9** indicates the compression ratio is the highest and processing speed the slowest.

For the custom-format archive, this option specifies the compression level of a single table data segment. By default, data is compressed at a medium level. Setting the non-zero compression level will result in that the entire text output files are to be compressed, as if the text has been compressed using the gzip tool, but the default method is non-compression. The .tar archive format does not support compression currently.
- **--lock-wait-timeout=TIMEOUT**
Do not keep waiting to obtain shared table locks at the beginning of the dump. Consider it as failed if you are unable to lock a table within the specified time. The timeout duration can be specified in any of the formats accepted by **SET statement_timeout**.
- **-, --help**
Shows help about **gs_dump** parameters and exits.

Dump parameters:

- **-a, --data-only**
Generates only the data, not the schema (data definition). Dumps the table data, big objects, and sequence values.
- **-b, --blobs**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **-c, --clean**
Before writing the command of creating database objects into the backup file, write the command of clearing (deleting) database objects to the backup files. (If no objects exist in the target database, **gs_restore** probably displays some error information.)
This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.
- **-C, --create**
The backup file content starts with the commands of creating the database and connecting to the created database. (If the script is in this format, any database to be connected is allowed before running the script.)
This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.
- **-E, --encoding=ENCODING**
Creates a dump file in the specified character set encoding. By default, the dump file is created in the database encoding. (Alternatively, you can set the environment variable **PGCLIENTENCODING** to the required dump encoding.)

- **-n, --schema=SCHEMA**

Dumps only schemas matching the schema names. This option contains the schema and all its contained objects. If this option is not specified, all non-system schemas in the target database will be dumped. Multiple schemas can be selected by specifying multiple **-n** options. The schema parameter is interpreted as a pattern according to the same rules used by the **\d** command of **gsqsl**. Therefore, multiple schemas can also be selected by writing wildcard characters in the pattern. When you use wildcards, quote the pattern to prevent the shell from expanding the wildcards.

 NOTE

- If **-n** is specified, **gs_dump** does not dump any other database objects that the selected schemas might depend upon. Therefore, there is no guarantee that the results of a specific-schema dump can be automatically restored to an empty database.
- If **-n** is specified, the non-schema objects are not dumped.

Multiple schemas can be dumped. Entering **-n schemaname** multiple times dumps multiple schemas.

For example:

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -n sch1 -n sch2
```

In the preceding example, **sch1** and **sch2** are dumped.

- **-N, --exclude-schema=SCHEMA**

Does not dump any tables matching the table pattern. The pattern is interpreted according to the same rules as for **-n**. **-N** can be specified multiple times to exclude schemas matching any of the specified patterns.

When both **-n** and **-N** are specified, the schemas that match at least one **-n** option but no **-N** is dumped. If **-N** is specified and **-n** is not, the schemas matching **-N** are excluded from what is normally dumped.

Dump allows you to exclude multiple schemas during dumping.

Specifies **-N exclude schema name** to exclude multiple schemas while dumping.

For example:

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -N sch1 -N sch2
```

In the preceding example, **sch1** and **sch2** will be excluded during the dumping.

- **-o, --oids**

Dumps object identifiers (OIDs) as parts of the data in each table. Use this parameter if your application references the OID columns in some way (for example, in a foreign key constraint). If the preceding situation does not occur, do not use this parameter.

- **-O, --no-owner**

Do not output commands to set ownership of objects to match the original database. By default, **gs_dump** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** command to set ownership of created database objects. These statements will fail when the script is running unless it is started by a system administrator (or the same user that owns all of the objects in the script). To make a script that can be stored by any user and give the user ownership of all objects, specify **-O**.

This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- **-s, --schema-only**

Dumps only the object definition (schema) but not data.

- **-S, --sysadmin=NAME**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **-t, --table=TABLE**

Specifies a list of tables, views, sequences, or foreign tables to be dumped. You can use multiple **-t** parameters or wildcard characters to specify tables.

When using wildcards to specify dump tables, quote the pattern to prevent the shell from expanding the wildcards.

The **-n** and **-N** options have no effect when **-t** is used, because tables selected by using **-t** will be dumped regardless of those options, and non-table objects will not be dumped.

NOTE

The number of **-t** parameters must be less than or equal to 100.

If the number of **-t** parameters is greater than 100, you are advised to use the **--include-table-file** parameter to replace some **-t** parameters.

If **-t** is specified, **gs_dump** does not dump any other database objects that the selected tables might depend upon. Therefore, there is no guarantee that the results of a specific-table dump can be automatically restored to an empty database.

-t tablename only dumps visible tables in the default search path. **-t '*.tablename'** dumps *tablename* tables in all the schemas of the dumped database. **-t schema.table** dumps tables in a specific schema.

-t tablename does not export the trigger information from a table.

For example:

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -t schema1.table1 -t schema2.table2
```

In the preceding example, **schema1.table1** and **schema2.table2** are dumped.

- **--include-table-file=FILENAME**

Specifies the table file to be dumped.

- **-T, --exclude-table=TABLE**

Specifies a list of tables, views, sequences, or foreign tables not to be dumped. You can use multiple **-t** parameters or wildcard characters to specify tables.

When **-t** and **-T** are input, the object will be stored in **-t** list not **-T** table object.

For example:

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -T table1 -T table2
```

In the preceding example, **table1** and **table2** are excluded from the dumping.

- **--exclude-table-file=FILENAME**

Specifies the table file to be dumped.

 NOTE

Same as **--include-table-file**, the content format of this parameter is as follows:

schema1.table1

schema2.table2

...

- **-x, --no-privileges|--no-acl**

Prevents the dumping of access permissions (grant/revoke commands).

- **--binary-upgrade**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--binary-upgrade-usermap="USER1=USER2"**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--column-inserts|--attribute-inserts**

Exports data by running the **INSERT** command with explicit column names {INSERT INTO table (column, ...) VALUES ...}. This will cause a slow restoration. However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content.

- **--disable-dollar-quoting**

Disables the use of dollar sign (\$) for function bodies, and forces them to be quoted using the SQL standard string syntax.

- **--disable-triggers**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--exclude-table-data=TABLE**

Does not dump data that matches any of table patterns. The pattern is interpreted according to the same rules as for **-t**.

--exclude-table-data can be entered more than once to exclude tables matching any of several patterns. When the user needs the specified table definition rather than data in the table, this option is helpful.

To exclude data of all tables in the database, see **--schema-only**.

- **--inserts**

Dumps data when the **INSERT** statement (rather than **COPY**) is issued. This will cause a slow restoration.

However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content. The restoration may fail if you rearrange the column order. The **--column-inserts** option is unaffected against column order changes, though even slower.

- **--no-security-labels**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--no-tablespaces**

Does not issue commands to select tablespaces. All the objects will be created during the restoration process, no matter which tablespace is selected when using this option.

This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- **--no-unlogged-table-data**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--non-lock-table**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **--quote-all-identifiers**

Forcibly quotes all identifiers. This parameter is useful when you dump a database for migration to a later version, in which additional keywords may be introduced.

- **--section=SECTION**

Specifies dumped name sections (pre-data, data, or post-data).

- **--serializable-deferrable**

Uses a serializable transaction for the dump to ensure that the used snapshot is consistent with later database status. Perform this operation at a time point in the transaction flow, at which everything is normal. This ensures successful transaction and avoids serialization failures of other transactions, which requires serialization again.

This option has no benefits for disaster recovery. During the upgrade of the original database, load a database copy as a report or other shared read-only dump is helpful. The option does not exist, dump reveals a status which is different from the submitted sequence status of any transaction.

This option will make no difference if there are no active read-write transactions when **gs_dump** is started. If the read-write transactions are in active status, the dump start time will be delayed for an uncertain period.

- **--use-set-session-authorization**

Specifies that the standard SQL **SET SESSION AUTHORIZATION** command rather than **ALTER OWNER** is returned to ensure the object ownership. This makes dumping more standard. However, if a dump file contains objects that have historical problems, restoration may fail. A dump using **SET SESSION AUTHORIZATION** requires the system administrator rights, whereas **ALTER OWNER** requires lower permissions.

- **--with-encryption=AES128**

Specifies that dumping data needs to be encrypted using AES128.

- **--with-key=KEY**

Specifies that the key length of AES128 must be 16 bytes.

- **--include-nodes**

Includes the **TO NODE** or **TO GROUP** statement in the dumped **CREATE TABLE** or **CREATE FOREIGN TABLE** statement. This parameter is valid only for HDFS and foreign tables.

- **--include-extensions**

Includes extensions in the dump.

- **--include-depend-objs**

Includes information about the objects that depend on the specified object in the backup result. This parameter takes effect only if the **-t** or **--include-table-file** parameter is specified.

- **--exclude-self**

Excludes information about the specified object from the backup result. This parameter takes effect only if the **-t** or **--include-table-file** parameter is specified.

- **--dont-overwrite-file**

The existing files in plain-text, .tar, and custom formats will be overwritten. This parameter is not used for the directory format.

For example:

Assume that the **backup.sql** file exists in the current directory. If you specify **-f backup.sql** in the input command, and the **backup.sql** file is generated in the current directory, the original file will be overwritten.

If the backup file already exists and **--dont-overwrite-file** is specified, an error will be reported with the message that the dump file exists.

```
gs_dump -p port_number postgres -f backup.sql -F plain --dont-overwrite-file
```

NOTE

- The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
- The **-c/--clean** and **-a/--data-only** parameters do not coexist.
- **--inserts/--column-inserts** and **-o/--oids** do not coexist, because **OIDs** cannot be set using the **INSERT** statement.
- **--role** must be used in conjunction with **--rolepassword**.
- **--binary-upgrade-usermap** must be used in conjunction with **--binary-upgrade**.
- **--include-depend-objs/--exclude-self** takes effect only when **-t/--include-table-file** is specified.
- **--exclude-self** must be used with **--include-depend-objs**.

Connection parameters:

- **-h, --host=HOSTNAME**

Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default is taken from the **PGHOST** environment variable (if available). Otherwise, a Unix domain socket connection is attempted.

This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.

Example: the host name

Environment Variable: **PGHOST**

- **-p, --port=PORT**

Specifies the host port.

Environment variable: **PGPORT**

- **-U, --username=NAME**

Specifies the user name of the host to connect to.

Environment variable: *PGUSER*

- **-w, --no-password**
Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.
- **-W, --password=PASSWORD**
Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** option. If the **-W** option is not provided and you are not a system administrator, the Dump Restore tool will ask you to enter a password.
- **--role=ROLENAME**
Specifies a role name to be used for creating the dump. If this option is selected, the **SET ROLE** command will be issued after the database is connected to **gs_dump**. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_dump**. It allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as a system administrator. This option allows dumping data without violating the policy.
- **--rolepassword=ROLEPASSWORD**
Password for the role

Description

Scenario 1

If your database cluster has any local additions to the template1 database, restore the output of **gs_dump** into an empty database with caution. Otherwise, you are likely to obtain errors due to duplicate definitions of the added objects. To create an empty database without any local additions, copy data from template0 rather than template1. Example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

The .tar format file size must be smaller than 8 GB. (This is the tar file format limitations.) The total size of a .tar archive and any of the other output formats are not limited, except possibly by the OS.

The dump file generated by **gs_dump** does not contain the statistics used by the optimizer to make execution plans. Therefore, you are advised to run **ANALYZE** after restoring from a dump file to ensure optimal performance. The dump file does not contain any **ALTER DATABASE ... SET** commands; these settings are dumped by **gs_dumpall**, along with database users and other installation settings.

Scenario 2

When the value of **SEQUENCE** reaches the maximum or minimum value, backing up the value of **SEQUENCE** using **gs_dump** will exit due to an execution error. Handle the problem by referring to the following example:

1. The value of **SEQUENCE** reaches the maximum value, but the maximum value is less than **2^63-2**.

Error message example:

Object defined by sequence

```
CREATE SEQUENCE seq INCREMENT 1 MINVALUE 1 MAXVALUE 3 START WITH 1;
```

Perform the **gs_dump** backup.

```
gs_dump -U dbadmin -W Bigdata@123 -p 37300 postgres -t PUBLIC.seq -f backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: The total objects number is 337.
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: WARNING: get invalid xid from GTM because
connection is not established
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: WARNING: Failed to receive GTM rollback
transaction response for aborting prepared (null).
gs_dump: [port='37300'] [postgres] [archiver (db)] [2019-12-27 15:09:49] query failed: ERROR: Can not
connect to gtm when getting gxid, there is a connection error.
gs_dump: [port='37300'] [postgres] [archiver (db)] [2019-12-27 15:09:49] query was: RELEASE bfnxtval
```

Handling procedure:

Run the following SQL statement to connect to the PostgreSQL database and change the maximum value of **sequence seq1**:

```
gsql -p 37300 postgres -r -c "ALTER SEQUENCE PUBLIC.seq MAXVALUE 10;"
```

Use the dump tool to back up the data.

```
gs_dump -U dbadmin -W Bigdata@123 -p 37300 postgres -t PUBLIC.seq -f backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: The total objects number is 337.
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: [100.00%] 337 objects have been dumped.
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: dump database postgres successfully
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: total time: 230 ms
```

2. The value of **SEQUENCE** reaches the minimum or the maximum value of **2⁶³-2**.

The **gs_dump** command does not support backup of the **SEQUENCE** value in this scenario.

NOTE

The SQL end does not support the modification of **MAXVALUE** when **SEQUENCE** reaches the maximum value of **2⁶³-2** or the modification of **MINVALUE** when **SEQUENCE** reaches the minimum value.

Examples

Use **gs_dump** to dump a database as a SQL text file or a file in other formats.

In the following examples, **Bigdata@123** indicates the password of the database user. **backup/MPPDB_backup.sql** indicates an exported file where **backup** indicates the relative path of the current directory. **37300** indicates the port ID of the database server. **postgres** indicates the name of the database to be accessed.

NOTE

Before exporting files, ensure that the directory exists and you have the read and write permissions on the directory.

Example 1: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.sql** file is in plain-text format.

```
gs_dump -U dbadmin -W Bigdata@123 -f backup/MPPDB_backup.sql -p 37300 postgres -F p
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: The total objects number is 356.
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: [100.00%] 356 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: total time: 1274 ms
```

Use **gsql** to import data from the export plain-text file.

Example 2: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.tar** file is in .tar format.

```
gs_dump -U dbadmin -W Bigdata@123 -f backup/MPPDB_backup.tar -p 37300 postgres -F t
gs_dump[port='37300'][postgres][2018-06-27 10:02:24]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: total time: 50086 ms
```

Example 3: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.dmp** file is in custom format.

```
gs_dump -U dbadmin -W Bigdata@123 -f backup/MPPDB_backup.dmp -p 37300 postgres -F c
gs_dump[port='37300'][postgres][2018-06-27 10:05:40]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: total time: 36620 ms
```

Example 4: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup** file is in directory format.

```
gs_dump -U dbadmin -W Bigdata@123 -f backup/MPPDB_backup -p 37300 postgres -F d
gs_dump[port='37300'][postgres][2018-06-27 10:16:04]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: total time: 33977 ms
```

Example 5: Use **gs_dump** to export the information of the **postgres** database, excluding the information of the table specified in the **/home/MPPDB_temp.sql** file. The exported **MPPDB_backup.sql** file is in plain-text format.

```
gs_dump -U dbadmin -W Bigdata@123 -p 37300 postgres --exclude-table-file=/home/MPPDB_temp.sql -f
backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2018-06-27 10:37:01]: The total objects number is 1367.
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: [100.00%] 1367 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: total time: 37017 ms
```

Example 6: Use **gs_dump** to export only the information about the views that depend on the **testtable** table. Create another **testtable** table, and then restore the views that depend on it.

Back up only the views that depend on the **testtable** table.

```
gs_dump -s -p 37300 postgres -t PUBLIC.testtable --include-depend-objs --exclude-self -f backup/
MPPDB_backup.sql -F p
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: The total objects number is 331.
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: [100.00%] 331 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: total time: 327 ms
```

Change the name of the **testtable** table.

```
gsql -p 37300 postgres -r -c "ALTER TABLE PUBLIC.testtable RENAME TO testtable_bak;"
```

Create a **testtable** table.

```
CREATE TABLE PUBLIC.testtable(a int, b int, c int);
```

Restore the views for the new **testtable** table.

```
gsql -p 37300 postgres -r -f backup/MPPDB_backup.sql
```

Helpful Links

[gs_dumpall](#)

20.2 gs_dumpall

Context

gs_dumpall, provided by GaussDB(DWS) is used to export all database information, including the data of the default **postgres** database, data of user-specified databases, and global objects of all databases in a cluster.

When **gs_dumpall** is used to export data, other users still can access the databases (readable or writable) in a cluster.

gs_dumpall can export complete, consistent data. For example, if **gs_dumpall** is started to export all databases from a cluster at T1, data of the databases at that time point will be exported, and modifications on the databases after that time point will not be exported.

To export all databases in a cluster:

- **gs_dumpall** exports all global objects, including information about database users and groups, tablespaces, and attributes (for example, global access permissions).
- **gs_dumpall** invokes **gs_dump** to export SQL scripts, which contain all the SQL statements required to restore databases.

The exported files are both plain-text SQL scripts. Use **gsql** to execute them to restore databases.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- To ensure the data consistency and integrity, **gs_dumpall** sets a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, **gs_dumpall** locks the table after it is released. If the table cannot be locked in the specified time, the dump will fail. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.
- During an export, **gs_dumpall** reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When using **gsql** to execute the SQL scripts, you need the administrator permissions so that you can add users and user groups, and create databases.

Syntax

```
gs_dumpall [OPTION]...
```

Parameter Description

Common parameters:

- **-f, --filename=FILENAME**
Sends the output to the specified file. If this parameter is omitted, the standard output is used.
- **-v, --verbose**
Specifies the verbose mode. This will cause **gs_dumpall** to output detailed object comments and start/stop times to the dump file, and progress messages to standard error.
- **-V, --version**
Prints the version information for **gs_dumpall** and exits.
- **--lock-wait-timeout=TIMEOUT**
Do not keep waiting to obtain shared table locks at the beginning of the dump. Consider it as failed if you are unable to lock a table within the specified time. The timeout duration can be specified in any of the formats accepted by **SET statement_timeout**.
- **-, --help**
Shows help about the command line parameters for **gs_dumpall** and exits.

Dump parameters:

- **-a, --data-only**
Dumps only the data, not the schema (data definition).
- **-c, --clean**
Runs SQL statements to delete databases before rebuilding them. Statements for dumping roles and tablespaces are added.
- **-g, --globals-only**
Dumps only global objects (roles and tablespaces) but no databases.
- **-o, --oids**
Dumps object identifiers (OIDs) as parts of the data in each table. Use this parameter if your application references the OID columns in some way (for example, in a foreign key constraint). If the preceding situation does not occur, do not use this parameter.
- **-O, --no-owner**
Do not output commands to set ownership of objects to match the original database. By default, **gs_dumpall** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** statement to set ownership of created schema elements. These statements will fail when the script is running unless it is started by a system administrator (or the same user that owns all of the objects in the script). To make a script that can be stored by any user and give the user ownership of all objects, specify **-O**.
- **-r, --roles-only**
Dumps only roles but not databases or tablespaces.
- **-s, --schema-only**
Dumps only the object definition (schema) but not data.
- **-S, --sysadmin=NAME**
Name of the system administrator during the dump.
- **-t, --tablespaces-only**

Dumps only tablespaces but not databases or roles.

- **-x, --no-privileges**
Prevents the dumping of access permissions (grant/revoke commands).
- **--column-inserts|--attribute-inserts**
Exports data by running the **INSERT** command with explicit column names {**INSERT INTO** table (column, ...) **VALUES** ...}. This will cause a slow restoration. However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content.
- **--disable-dollar-quoting**
Disables the use of dollar sign (\$) for function bodies, and forces them to be quoted using the SQL standard string syntax.
- **--disable-triggers**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--inserts**
Dumps data when the **INSERT** statement (rather than **COPY**) is issued. This will cause a slow restoration. The restoration may fail if you rearrange the column order. The **--column-inserts** parameter is safer against column order changes, though even slower.
- **--no-security-labels**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--no-tablespaces**
Do not output statements to create tablespaces or select tablespaces for objects. All the objects will be created during the restoration process, no matter which tablespace is selected when using this option.
- **--no-unlogged-table-data**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--quote-all-identifiers**
Forcibly quotes all identifiers. This parameter is useful when you dump a database for migration to a later version, in which additional keywords may be introduced.
- **--dont-overwrite-file**
Do not overwrite the current file.
- **--use-set-session-authorization**
Specifies that the standard SQL **SET SESSION AUTHORIZATION** command rather than **ALTER OWNER** is returned to ensure the object ownership. This makes dumping more standard. However, if a dump file contains objects that have historical problems, restoration may fail. A dump using **SET SESSION AUTHORIZATION** requires the system administrator rights, whereas **ALTER OWNER** requires lower permissions.
- **--with-encryption=AES128**
Specifies that dumping data needs to be encrypted using AES128.

- **--with-key=KEY**
Specifies that the key length of AES128 must be 16 bytes.
- **--include-extensions**
Backs up all CREATE EXTENSION statements if the **include-extensions** parameter is set.
- **--include-templatedb**
Includes template databases during the dump.
- **--dump-nodes**
Includes nodes and Node Groups during the dump.
- **--include-nodes**
Includes the **TO NODE** statement in the dumped **CREATE TABLE** statement.
- **--include-buckets**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--dump-wrm**
Includes workload resource manager (resource pool, load group, and load group mapping) during the dump.
- **--binary-upgrade**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--binary-upgrade-usermap="USER1=USER2"**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--tablespaces-postfix**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--parallel-jobs**
Specifies the number of concurrent backup processes. The value range is 1-1000.

 NOTE

- The **-g/--globals-only** and **-r/--roles-only** parameters do not coexist.
- The **-g/--globals-only** and **-t/--tablespaces-only** parameters do not coexist.
- The **-r/--roles-only** and **-t/--tablespaces-only** parameters do not coexist.
- The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
- The **-r/--roles-only** and **-a/--data-only** parameters do not coexist.
- The **-t/--tablespaces-only** and **-a/--data-only** parameters do not coexist.
- The **-g/--globals-only** and **-a/--data-only** parameters do not coexist.
- **--tablespaces-postfix** must be used in conjunction with **--binary-upgrade**.
- **--binary-upgrade-usermap** must be used in conjunction with **--binary-upgrade**.
- **--parallel-jobs** must be used in conjunction with **-f/--file**.

Connection parameters:

- **-h, --host**

Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default value is taken from the *PGHOST* environment variable. If it is not set, a UNIX domain socket connection is attempted.

This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.

Environment Variable: *PGHOST*

- -l, --database

Specifies the name of the database connected to dump all objects and discover other databases to be dumped. If this parameter is not specified, the **postgres** database will be used. If the **postgres** database does not exist, **template1** will be used.

- -p, --port

Specifies the TCP port listened to by the server or the local UNIX domain socket file name extension to ensure a correct connection. The default value is the *PGPORT* environment variable.

Environment variable: *PGPORT*

- -U, --username

Specifies the user name to connect to.

Environment variable: *PGUSER*

- -w, --no-password

Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.

- -W, --password

Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** option. If the **-W** option is not provided and you are not a system administrator, the Dump Restore tool will ask you to enter a password.

- --role

Specifies a role name to be used for creating the dump. This option causes **gs_dumpall** to issue the **SET ROLE** statement after connecting to the database. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_dumpall**. It allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as a system administrator. This option allows dumping data without violating the policy.

- --rolepassword

Specifies the password of the specific role.

Description

The **gs_dumpall** internally invokes **gs_dump**. For details about the diagnosis information, see **gs_dump**.

Once **gs_dumpall** is restored, run ANALYZE on each database so that the optimizer can provide useful statistics.

gs_dumpall requires all needed tablespace directories to exist before the restoration. Otherwise, database creation will fail if the databases are in non-default locations.

Examples

Run **gs_dumpall** to export all databases from a cluster at a time.

NOTE

gs_dumpall supports only plain-text format export. Therefore, only **gs_sql** can be used to restore a file exported using **gs_dumpall**.

```
gs_dumpall -f backup/bkp2.sql -p 37300
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:09]: The total objects number is 2371.
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:35]: [100.00%] 2371 objects have been
dumped.
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:46]: dump database dbname='postgres'
successfully
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:46]: total time: 55567 ms
gs_dumpall[port='37300'][2018-06-27 09:55:46]: dumpall operation successful
gs_dumpall[port='37300'][2018-06-27 09:55:46]: total time: 56088 ms
```

Helpful Links

[gs_dump](#)

20.3 gs_restore

Context

gs_restore, provided by GaussDB(DWS), is used to import data that was exported using **gs_dump**. It can also be used to import files exported by **gs_dump**.

It has the following functions:

- Imports data to the database.
If a database is specified, data is imported in the database. For parallel import, the password for connecting to the database is required.
- Imports data to the script file.
If the database storing imported data is not specified, a script containing the SQL statement to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output format of **gs_dump**.

Syntax

```
gs_restore [OPTION]... FILE
```

 NOTE

- The **FILE** does not have a short or long parameter. It is used to specify the location for the archive files.
- The **dbname** or **-l** parameter is required as prerequisites. Users cannot enter **dbname** and **-l** parameters at the same time.
- **gs_restore** incrementally imports data by default. To prevent data exception caused by multiple import jobs, use the **-e** and **-c** parameters during the jobs. In this way, existing database objects in a target database are deleted before import; and errors during import will be ignored to proceed the import and the error information will be displayed after the import.

Parameter Description

Common parameters:

- **-d, --dbname=NAME**
Connects to the **dbname** database and imports data to the database.
- **-f, --file=FILENAME**
Specifies the output file for the generated script, or uses the output file in the list specified using **-l**.
The default is the standard output.

 NOTE

- **-f** cannot be used in conjunction with **-d**.
- **-F, --format=c|d|t**
Specifies the format of the archive. The format does not need to be specified because the *gs_restore* determines the format automatically.
Value range:
 - **c/custom**: The archive form is the customized format in section 4.21-*gs_dump*.
 - **d/directory**: The archive form is a directory archive format.
 - **t/tar**: The archive form is a tar archive format.
- **-l, --list**
Lists the forms of the archive. The operation output can be used for the input of the **-L** parameter. If filtering parameters, such as **-n** or **-t**, are used together with **-l**, they will restrict the listed items.
- **-v, --verbose**
Specifies the verbose mode.
- **-V, --version**
Prints the **gs_restore** version and exits.
- **-, --help**
Shows help information about the parameters of **gs_restore** and exits.

Import parameters

- **-a, -data-only**
Imports only the data, not the schema (data definition). **gs_restore** incrementally imports data.

- **-c, --clean**
Cleans (deletes) existing database objects in the database to be restored before recreating them.
- **-C, --create**
Creates the database before importing data to it. (When this parameter is used, the database named with **-d** is used to issue the initial **CREATE DATABASE** command. All data is imported to the database that appears in the archive files.)
- **-e, --exit-on-error**
Exits if an error occurs when you send the SQL statement to the database. If you do not exit, the commands will still be sent and error information will be displayed when the import ends.
- **-I, --index=NAME**
Imports only the definition of the specified index. Multiple indexes can be imported. Enter **-I index** multiple times to import multiple indexes.
For example:

```
gs_restore -h host_name -p port_number -d postgres -I Index1 -I Index2 backup/MPPDB_backup.tar
```


In this example, *Index1* and *Index2* will be imported.
- **-j, --jobs=NUM**
Specifies the number of concurrent, the most time-consuming jobs of **gs_restore** (such as loading data, creating indexes, or creating constraints). This parameter can greatly reduce the time to import a large database to a server running on a multiprocessor machine.
Each job is one process or one thread, depending on the OS; and uses a separate connection to the server.
The optimal value for this parameter depends on the hardware settings of the server, client, and network. Factors include the number of CPU cores and disk settings. It is recommended that the parameter be set to the number of CPU cores on the server. In addition, a larger value can also lead to faster import in many cases. However, an overly large value will lead to decreased performance because of thrashing.
This parameter supports custom-format archives only. The input file must be a regular file (not the pipe file). This parameter can be ignored when you select the script method rather than connect to a database server. In addition, multiple jobs cannot be used in conjunction with the **--single-transaction** parameter.
- **-L, --use-list=FILENAME**
Imports only archive elements that are listed in **list-file** and imports them in the order that they appear in the file. If filtering parameters, such as **-n** or **-t**, are used in conjunction with **-L**, they will further limit the items to be imported.
list-file is normally created by editing the output of a previous **-I** parameter. File lines can be moved or removed, and can also be commented out by placing a semicolon (;) at the beginning of the row. An example is provided in this document.
- **-n, --schema=NAME**
Restores only objects that are listed in schemas.

This parameter can be used in conjunction with the **-t** parameter to import a specific table.

Entering **-n schemaname** multiple times can import multiple schemas.

For example:

```
gs_restore -h host_name -p port_number -d postgres -n sch1 -n sch2 backup/MPPDB_backup.tar
```

In this example, **sch1** and **sch2** will be imported.

- **-O, --no-owner**

Do not output commands to set ownership of objects to match the original database. By default, **gs_restore** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** statement to set ownership of created schema elements. Unless the system administrator or the user who has all the objects in the script initially accesses the database. Otherwise, the statement will fail. Any user name can be used for the initial connection using **-O**, and this user will own all the created objects.

- **-P, --function=NAME(args)**

Imports only listed functions. You need to correctly spell the function name and the parameter based on the contents of the dump file in which the function exists.

Entering **-P** alone means importing all function-name(args) functions in a file. Entering **-P** with **-n** means importing the function-name(args) functions in a specified schema. Entering **-P** multiple times and using **-n** once means that all imported functions are in the **-n** schema by default.

You can enter **-n schema-name -P 'function-name(args)'** multiple times to import functions in specified schemas.

For example:

```
./gs_restore -h host_name -p port_number -d postgres -n test1 -P 'Func1(integer)' -n test2 -P 'Func2(integer)' backup/MPPDB_backup.tar
```

In this example, both **Func1 (i integer)** in the **test1** schema and **Func2 (j integer)** in the **test2** schema will be imported.

- **-s, --schema-only**

Imports only schemas (data definitions), instead of data (table content). The current sequence value will not be imported.

- **-S, --sysadmin=NAME**

Specifies a reserved port for function expansion. This parameter is not recommended.

- **-t, --table=NAME**

Imports only listed table definitions or data, or both. This parameter can be used in conjunction with the **-n** parameter to specify a table object in a schema. When **-n** is not entered, the default schema is **PUBLIC**. Entering **-n schemaname -t tablename** multiple times can import multiple tables in a specified schema.

For example:

Import **table1** in the **PUBLIC** schema.

```
gs_restore -h host_name -p port_number -d postgres -t table1 backup/MPPDB_backup.tar
```

Import **test1** in the **test1** schema and **test2** in the **test2** schema.

```
gs_restore -h host_name -p port_number -d postgres -n test1 -t test1 -n test2 -t test2 backup/MPPDB_backup.tar
```

Import **table1** in the **PUBLIC** schema and **test1** in the **test1** schema.

```
gs_restore -h host_name -p port_number -d postgres -n PUBLIC -t table1 -n test1 -t table1 backup/  
MPPDB_backup.tar
```

NOTICE

-t does not support the **schema_name.table_name** input format.

- **-T, --trigger=NAME**
This parameter is reserved for extension.
- **-x, --no-privileges/--no-acl**
Prevents the import of access permissions (**GRANT/REVOKE** commands).
- **-1, --single-transaction**
Executes import as a single transaction (that is, commands are wrapped in **BEGIN/COMMIT**).
This parameter ensures that either all the commands are completed successfully or no application is changed. This parameter means **--exit-on-error**.
- **--disable-triggers**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--no-data-for-failed-tables**
By default, table data will be imported even if the statement to create a table fails (for example, the table already exists). Data in such table is skipped using this parameter. This operation is useful if the target database already contains the desired table contents.
This parameter takes effect only when you import data directly into a database, not when you output SQL scripts.
- **--no-security-labels**
Specifies a reserved port for function expansion. This parameter is not recommended.
- **--no-tablespaces**
Does not issue commands to select tablespaces. If this parameter is used, all objects will be created during the import process no matter which tablespace is selected.
- **--section=SECTION**
Imports the listed sections (such as pre-data, data, or post-data).
- **--use-set-session-authorization**
Is used for plain-text backup.
Outputs the **SET SESSION AUTHORIZATION** statement instead of the **ALTER OWNER** statement to determine object ownership. This parameter makes dump more standards-compatible. If the records of objects in exported files are referenced, import may fail. Only administrators can use the **SET SESSION AUTHORIZATION** statement to dump data, and the administrators must manually change and verify the passwords of exported files by referencing the **SET SESSION AUTHORIZATION** statement before import. The **ALTER OWNER** statement requires lower permissions.

- **--with-key=KEY**
Specifies that the key length of AES128 must be 16 bytes.

 NOTE

If the dump is encrypted, enter the **--with-key <keyname>** parameter in the **gs_restore** command. If it is not entered, you will receive an error message.
Enter the same key while entering the dump.

NOTICE

- If any local additions need to be added to the template1 database during the installation, restore the output of **gs_restore** into an empty database with caution. Otherwise, you are likely to obtain errors due to duplicate definitions of the added objects. To create an empty database without any local additions, copy data from template0 rather than template1. Example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

- **gs_restore** cannot import large objects selectively. For example, it can only import the objects of a specified table. If an archive contains large objects, all large objects will be imported, or none of them will be restored if they are excluded by using **-L**, **-t**, or other parameters.

 NOTE

1. The **-d/--dbname** and **-f/--file** parameters do not coexist.
2. The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
3. The **-c/--clean** and **-a/--data-only** parameters do not coexist.
4. When **--single-transaction** is used, **-j/--jobs** must be a single job.
5. **--role** must be used in conjunction with **--rolepassword**.

Connection parameters:

- **-h, --host=HOSTNAME**
Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default value is taken from the *PGHOST* environment variable. If it is not set, a UNIX domain socket connection is attempted.
This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.
- **-p, --port=PORT**
Specifies the TCP port listened to by the server or the local UNIX domain socket file name extension to ensure a correct connection. The default value is the *PGPORT* environment variable.
- **-U, --username=NAME**
Specifies the user name to connect to.
- **-w, --no-password**
Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.

- **-W, --password=PASSWORD**
Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** parameter. If the **-W** parameter is not provided and you are not a system administrator, **gs_restore** will ask you to enter a password.
- **--role=ROLENAME**
Specifies a role name for the import operation. If this parameter is selected, the **SET ROLE** statement will be issued after **gs_restore** connects to the database. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_restore**. This parameter allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as the initial user. This parameter allows data to be imported without violating the policy.
- **--rolepassword=ROLEPASSWORD**
Specifies the password of the specific role.

Examples

Special case: Execute the **gsql** tool. Run the following commands to import the **MPPDB_backup.sql** file in the export folder (in plain-text format) generated by **gs_dump/gs_dumpall** to the **postgres** database:

```
gsq| -d postgres -p 8000 -W Bigdata@123 -f /home/omm/test/MPPDB_backup.sql
SET
SET
SET
SET
SET
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
CREATE INDEX
SET
CREATE INDEX
REVOKE
REVOKE
GRANT
GRANT
total time: 30476 ms
```

gs_restore is used to import the files exported by **gs_dump**.

Example 1: Execute the **gs_restore** tool to import the exported **MPPDB_backup.dmp** file (custom format) to the **postgres** database.

```
gs_restore -W Bigdata@123 backup/MPPDB_backup.dmp -p 8000 -d postgres
gs_restore: restore operation successful
gs_restore: total time: 13053 ms
```

Example 2: Execute the **gs_restore** tool to import the exported **MPPDB_backup.tar** file (.tar format) to the **postgres** database.

```
gs_restore backup/MPPDB_backup.tar -p 8000 -d postgres
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21203 ms
```

Example 3: Execute the **gs_restore** tool to import the exported **MPPDB_backup** file (directory format) to the **postgres** database.

```
gs_restore backup/MPPDB_backup -p 8000 -d postgres
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21003 ms
```

Example 4: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import all the object definitions and data in the **PUBLIC** schema. Existing objects in the target database are deleted before the import. Objects having cross-schema dependency needs to be manually deleted.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d postgres -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR: cannot drop table table1 because other objects
depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

Manually delete the dependency and recreate it after the import is complete.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d postgres -e -c -n PUBLIC
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 2203 ms
```

Example 5: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import only the definition of **table1** in the **PUBLIC** schema.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d postgres -e -c -s -n PUBLIC -t table1
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21000 ms
```

Example 6: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import only the data of **table1** in the **PUBLIC** schema.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d postgres -e -a -n PUBLIC -t table1
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 20203 ms
```

Helpful Links

[gs_dump](#), [gs_dumpall](#)

21 Glossary

Term	Description
A - E	
ACID	Atomicity, Consistency, Isolation, and Durability (ACID). These are a set of properties of database transactions in a DBMS.
cluster ring	A cluster ring consists of several physical servers. The primary-standby-secondary relationships among its DN's do not involve external DN's. That is, none of the primary, standby, or secondary counterparts of DN's belonging to the ring are deployed in other rings. A ring is the smallest unit used for scaling.
Bgwriter	A background write thread created when the database starts. The thread pushes dirty pages in the database to a permanent device (such as a disk).
bit	The smallest unit of information handled by a computer. One bit is expressed as a 1 or a 0 in a binary numeral, or as a true or a false logical condition. A bit is physically represented by an element such as high or low voltage at one point in a circuit, or a small spot on a disk that is magnetized in one way or the other. A single bit conveys little information a human would consider meaningful. A group of eight bits, however, makes up a byte, which can be used to represent many types of information, such as a letter of the alphabet, a decimal digit, or other character.
Bloom filter	Bloom filter is a space-efficient binary vectorized data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, in other words, a query returns either "possibly in set (possible error)" or "definitely not in set". In the cases, Bloom filter sacrificed the accuracy for time and space.

Term	Description
CCN	Control Coordinator (CCN) centrally, dynamically manages load in GaussDB(DWS). It is responsible for determining, queuing, and scheduling complex operations in each CN for dynamic load management.
CIDR	Classless Inter-Domain Routing (CIDR). Whereas classical network design for IPv4 sized the network prefix as one or more 8-bit groups, resulting in the blocks of Class A (8-bit), B (16-bit), or C (24-bit) addresses, CIDR allocates address space on any address bit boundary. A CIDR address is in the format of <i>IP address/Number of bits in a network ID</i> . For example, in 192.168.23.35/21 , 21 indicates that the first 21 bits are the network prefix and others are the host ID.
Cgroup	A control group (Cgroup), also called a priority group (PG) in GaussDB(DWS). The Cgroup is a kernel feature of SUSE Linux and Red Hat that can limit, account for, and isolate the resource usage of a collection of processes.
CLI	Command-line interface (CLI). Users use the CLI to interact with applications. Its input and output are based on texts. Commands are entered through keyboards or similar devices and are compiled and executed by applications. The results are displayed in text or graphic forms on the terminal interface.
CM	Cluster Manager (CM) manages and monitors the running status of functional units and physical resources in the distributed system, ensuring stable running of the entire system.
CMS	The Cluster Management Service (CMS) component manages the cluster status.
CN	The Coordinator (CN) stores database metadata, splits query tasks and supports their execution, then compiles the query results returned from DNs.
CU	Compression Unit (CU) is the smallest storage unit in a column-storage table.
core file	A file that is created when memory overwriting, assertion failures, or access to invalid memory occurs in a process, causing it to fail. This file is then used for further analysis. A core file contains a memory dump, in an all-binary and port-specific format. The name of a core file consists of the word "core" and the OS process ID. The core file is available regardless of the type of platform.

Term	Description
core dump	When a program stops abnormally, the core dump, memory dump, or system dump records the state of the working memory of the program at that point in time. In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and OS flags and information. A core dump is often used to assist diagnosis and computer program debugging.
DBA	A database administrator (DBA) instructs or executes database maintenance operations.
DBLINK	An object defining the path from one database to another. A remote database object can be queried with DBLINK.
DBMS	Database Management System (DBMS) is a piece of system management software that allows users to access information in a database. This is a collection of programs that allows you to access, manage, and query data in a database. A DBMS can be classified as memory DBMS or disk DBMS based on the location of the data.
DCL	Data control language (DCL)
DDL	Data definition language (DDL)
DML	Data manipulation language (DML)
DN	Datanode performs table data storage and query operations.
ETCD	The Editable Text Configuration Daemon (ETCD) is a distributed key-value storage system used for configuration sharing and service discovery (registration and search).
ETL	Extract-Transform-Load (ETL) refers to the process of data transmission from the source to the target database.
backup	A backup, or the process of backing up, refers to the copying and archiving of computer data in case of data loss.
backup and restoration	A collection of concepts, procedures, and strategies to protect data loss caused by invalid media or misoperations.
standby server	A node in the GaussDB(DWS) HA solution. It functions as a backup of the primary server. If the primary server is behaving abnormally, the standby server is promoted to primary, ensuring data service continuity.

Term	Description
crash	A crash (or system crash) is an event in which a computer or a program (such as a software application or an OS) ceases to function properly. Often the program will exit after encountering this type of error. Sometimes the offending program may appear to freeze or hang until a crash reporting service documents details of the crash. If the program is a critical part of the OS kernel, the entire computer may crash (possibly resulting in a fatal system error).
encoding	Encoding is representing data and information using code so that it can be processed and analyzed by a computer. Characters, digits, and other objects can be converted into digital code, or information and data can be converted into the required electrical pulse signals based on predefined rules.
encoding technology	A technology that presents data using a specific set of characters, which can be identified by computer hardware and software.
table	A set of columns and rows. Each column is referred to as a field. The value in each field represents a data type. For example, if a table contains people's names, cities, and states, it has three columns: Name , City , and State . In every row in the table, the Name column contains a name, the City column contains a city, and the State column contains a state.
tablespace	A tablespace is a logical storage structure that contains tables, indexes, large objects, and long data. A tablespace provides an abstract layer between physical data and logical data, and provides storage space for all database objects. When you create a table, you can specify which tablespace it belongs to.
concurrency control	A DBMS service that ensures data integrity when multiple transactions are concurrently executed in a multi-user environment. In a multi-threaded environment, GaussDB(DWS) concurrency control ensures that database operations are safe and all database transactions remain consistent at any given time.
query	Specifies requests sent to the database, such as updating, modifying, querying, or deleting information.
query operator	An iterator or a query tree node, which is a basic unit for the execution of a query. Execution of a query can be split into one or more query operators. Common query operators include scan, join, and aggregation.
query fragment	Each query task can be split into one or more query fragments. Each query fragment consists of one or more query operators and can independently run on a node. Query fragments exchange data through data flow operators.

Term	Description
durability	One of the ACID features of database transactions. Durability indicates that transactions that have been committed will permanently survive and not be rolled back.
stored procedure	A group of SQL statements compiled into a single execution plan and stored in a large database system. Users can specify a name and parameters (if any) for a stored procedure to execute the procedure.
OS	An operating system (OS) is loaded by a bootstrap program to a computer to manage other programs in the computer. applications on a computer or similar device.
secondary server	To ensure high cluster availability, the primary server synchronizes logs to the secondary server if data synchronization between the primary and standby servers fails. If the primary server suddenly breaks down, the standby server is promoted to primary and synchronizes logs from the secondary server for the duration of the breakdown.
BLOB	Binary large object (BLOB) is a collection of binary data stored in a database, such as videos, audio, and images.
segment	A segment in the database indicates a part containing one or more regions. Region is the smallest range of a database and consists of data blocks. One or more segments comprise a tablespace.
F – J	
failover	Automatic switchover from a faulty node to its standby node. Reversely, automatic switchback from the standby node to the primary node is called failback.
FDW	A foreign data wrapper (FDW) is a SQL interface provided by Postgres. It is used to access big data objects stored in remote data so that DBAs can integrate data from unrelated data sources and store them in public schema in the database.
freeze	An operation automatically performed by the AutoVacuum Worker process when transaction IDs are exhausted. GaussDB(DWS) records transaction IDs in row headings. When a transaction reads a row, the transaction ID in the row heading and the actual transaction ID are compared to determine whether this row is explicit. Transaction IDs are integers containing no symbols. If exhausted, transaction IDs are re-calculated outside of the integer range, causing the explicit rows to become implicit. To prevent such a problem, the freeze operation marks a transaction ID as a special ID. Rows marked with these special transaction IDs are explicit to all transactions.

Term	Description
GDB	As a GNU debugger, GDB allows you to see what is going on 'inside' another program while it executes or what another program was doing the moment that it crashed. GDB can perform four main kinds of things (make PDK functions stronger) to help you catch bugs in the act: • Starts your program, specifying anything that might affect its behavior. • Stops a program in a specific condition. • Checks what happens when a program stops. • Changes things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.
GDS	General Data Service (GDS). To import data to GaussDB(DWS), you need to deploy the tool on the server where the source data is stored so that DNs can use this tool to obtain data.
GNU	The GNU Project was publicly announced on September 27, 1983 by Richard Stallman, aiming at building an OS composed wholly of free software. GNU is a recursive acronym for "GNU's Not Unix!". Stallman announced that GNU should be pronounced as Guh-NOO. Technically, GNU is similar to Unix in design, a widely used commercial OS. However, GNU is free software and contains no Unix code.
gsql	An interactive GaussDB(DWS) terminal. It enables you to interactively type in queries, issue them to GaussDB(DWS), and view the query results. Queries can also be entered from files. gsql supports many meta commands and shell-like commands, allowing you to conveniently compile scripts and automate tasks.
GTM	Global Transaction Manager (GTM) manages the status of transactions.
GUC	Grand unified configuration (GUC) includes parameters for running databases, the values of which determine database system behavior.
HA	High availability (HA) is a solution in which two modules operate in primary/standby mode to achieve high availability. This solution helps to minimize the duration of service interruptions caused by routine maintenance (planned) or sudden system breakdowns (unplanned), improving the system and application usability.

Term	Description
HBA	Host-based authentication (HBA) allows hosts to authenticate on behalf of all or some of the system users. It can apply to all users on a system or a subset using the Match directive. This type of authentication can be useful for managing computing clusters and other fairly homogenous pools of machines. In all, three files on the server and one on the client must be modified to prepare for host-based authentication.
HDFS	Hadoop Distributed File System (HDFS) is a subproject of Apache Hadoop. HDFS is highly fault tolerant and is designed to run on low-end hardware. The HDFS provides high-throughput access to large data sets and is ideal for applications having large data sets.
server	A combination of hardware and software designed for providing clients with services. This word alone refers to the computer running the server OS, or the software or dedicated hardware providing services.
advanced package	Logical and functional stored procedures and functions provided by GaussDB(DWS).
isolation	One of the ACID features of database transactions. Isolation means that the operations inside a transaction and data used are isolated from other concurrent transactions. The concurrent transactions do not affect each other.
relational database	A database created using a relational model. It processes data using methods of set algebra.
archive thread	A thread started when the archive function is enabled on a database. The thread archives database logs to a specified path.
failover	The automatic substitution of a functionally equivalent system component for a failed one. The system component can be a processor, server, network, or database.
environment variable	An environment variable defines the part of the environment in which a process runs. For example, it can define the part of the environment as the main directory, command search path, terminal that is in use, or the current time zone.
checkpoint	A mechanism that stores data in the database memory to disks at a certain time. GaussDB(DWS) periodically stores the data of committed and uncommitted transactions to disks. The data and redo logs can be used for database restoration if a database restarts or breaks down.
encryption	A function hiding information content during data transmission to prevent the unauthorized use of the information.
node	Cluster nodes (or nodes) are physical and virtual servers that make up the GaussDB(DWS) cluster environment.

Term	Description
error correction	A technique that automatically detects and corrects errors in software and data streams to improve system stability and reliability.
process	An instance of a computer program that is being executed. A process may be made up of multiple threads of execution. Other processes cannot use a thread occupied by the process.
PITR	Point-In-Time Recovery (PITR) is a backup and restoration feature of GaussDB(DWS). Data can be restored to a specified point in time if backup data and WAL logs are normal.
record	In a relational database, a record corresponds to data in each row of a table.
cluster	A cluster is an independent system consisting of servers and other resources, ensuring high availability. In certain conditions, clusters can implement load balancing and concurrent processing of transactions.
K - O	
LLVM	LLVM is short for Low Level Virtual Machine. Low Level Virtual Machine (LLVM) is a compiler framework written in C++ and is designed to optimize the compile-time, link-time, run-time, and idle-time of programs that are written in arbitrary programming languages. It is open to developers and compatible with existing scripts.
LVS	Linux Virtual Server (LVS), a virtual server cluster system, is used for balancing the load of a cluster.
MPP	Massive Parallel Processing (MPP) refers to cluster architecture that consists of multiple machines. The architecture is also called a cluster system.
MVCC	Multi-Version Concurrency Control (MVCC) is a protocol that allows a tuple to have multiple versions, on which different query operations can be performed. A basic advantage is that read and write operations do not conflict.
NameNode	The NameNode is the centerpiece of a Hadoop file system, managing the namespace of the file system and client access to files.
OBS	Object Storage Service (OBS), an object-based cloud storage service.

Term	Description
OLAP	Online analytical processing (OLAP) is the most important application in the database warehouse system. It is dedicated to complex analytical operations, helps decision makers and executives to make decisions, and rapidly and flexibly processes complex queries involving a great amount of data based on analysts' requirements. In addition, the OLAP provides decision makers with query results that are easy to understand, allowing them to learn the operating status of the enterprise. These decision makers can then produce informed and accurate solutions based on the query results.
OM	Operations Management (OM) provides management interfaces and tools for routine maintenance and configuration management of the cluster.
ORC	Optimized Row Columnar (ORC) is a widely used file format for structured data in a Hadoop system. It was introduced from the Hadoop HIVE project.
client	A computer or program that accesses or requests services from another computer or program.
free space management	A mechanism for managing free space in a table. This mechanism enables the database system to record free space in each table and establish an easy-to-search data structure, accelerating operations (such as INSERT) performed on the free space.
junk tuple	A tuple that is deleted using the DELETE and UPDATE statements. When deleting a tuple, GaussDB(DWS) only marks the tuples that are to be cleared. The Vacuum thread will then periodically clear these junk tuples.
column	An equivalent concept of "field". A database table consists of one or more columns. Together they describe all attributes of a record in the table.
logical node	Multiple logical nodes can be installed on the same node. A logical node is a database instance.
schema	Collection of database objects, including logical structures, such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links.
schema file	A SQL file that determines the database structure.
P – T	
page	Smallest storage unit for row storage in the relational object structure in GaussDB(DWS). The default size of a page is 8 KB.
PostgreSQL	An open-source DBMS developed by volunteers all over the world. PostgreSQL is not controlled by any companies or individuals. Its source code can be used for free.

Term	Description
Postgres-XC	Postgres-XC is an open source PostgreSQL cluster to provide write-scalable, synchronous, multi-master PostgreSQL cluster solution.
Postmaster	A thread started when the database service is started. It listens to connection requests from other nodes in the cluster or from clients. After receiving and accepting a connection request from the standby server, the primary server creates a WAL Sender thread to interact with the standby server.
RHEL	Red Hat Enterprise Linux (RHEL)
redo log	A log that contains information required for performing an operation again in a database. If a database is faulty, redo logs can be used to restore the database to its original state.
SCTP	The Stream Control Transmission Protocol (SCTP) is a transport-layer protocol defined by Internet Engineering Task Force (IETF) in 2000. The protocol ensures the reliability of datagram transport based on unreliable service transmission protocols by transferring SCN narrowband signaling over IP network.
session	A task created by a database for a connection when an application attempts to connect to the database. Sessions are managed by the session manager. They execute initial tasks to perform all user operations.
shared-nothing architecture	A distributed computing architecture, in which none of the nodes share CPUs or storage resources. This architecture has good scalability.
SLES	SUSE Linux Enterprise Server (SLES) is an enterprise Linux OS provided by SUSE.
SMP	Symmetric multiprocessing (SMP) lets multiple CPUs run on a computer and share the same memory and bus. To ensure an SMP system achieves high performance, an OS must support multi-tasking and multi-thread processing. In databases, SMP means to concurrently execute queries using the multi-thread technology, efficiently using all CPU resources and improving query performance.
SQL	Structure Query Language (SQL) is a standard database query language. It consists of DDL, DML, and DCL.

Term	Description
SSL	Secure Socket Layer (SSL) is a network security protocol introduced by Netscape. SSL is a security protocol based on the TCP and IP communications protocols and uses the public key technology. SSL supports a wide range of networks and provides three basic security services, all of which use the public key technology. SSL ensures the security of service communication through the network by establishing a secure connection between the client and server and then sending data through this connection.
convergence ratio	Downlink to uplink bandwidth ratio of a switch. A high convergence ratio indicates a highly converged traffic environment and severe packet loss.
trace	A way of logging to record information about the way a program is executed. This information is typically used by programmers for debugging purposes. System administrators and technical support can diagnose common problems by using software monitoring tools and based on this information.
full backup	Backup of the entire database cluster.
full synchronization	A data synchronization mechanism specified in the GaussDB(DWS) HA solution. Used to synchronize all data from the primary server to a standby server.
log file	A file to which a computer system writes a record of its activities.
transaction	A logical unit of work performed within a DBMS against a database. A transaction consists of a limited database operation sequence, and must have ACID features.
data	A representation of facts or directives for manual or automatic communication, explanation, or processing. Data includes constants, variables, arrays, and strings.
data redistribution	A mode in which table data is split and stored on each database instance in a distributed system. Table data can be distributed in hash, replication, or random mode. In hash mode, a hash value is calculated based on the value of a specified column in a tuple, and then the target storage location of the tuple is determined based on the mapping between nodes and hash values. In replication mode, tuples are replicated to all nodes. In random mode, data is randomly distributed to the nodes.

Term	Description
data distribution	A mode in which table data is split and stored on each database instance in a distributed system. Table data can be distributed in hash, replication, or random mode. In hash mode, a hash value is calculated based on the value of a specified column in a tuple, and then the target storage location of the tuple is determined based on the mapping between nodes and hash values. In replication mode, tuples are replicated to all nodes. In random mode, data is randomly distributed to the nodes.
data partitioning	A division of a logical database or its constituent elements into multiple parts (partitions) whose data does not overlap based on specified ranges. Data is mapped to storage locations based on the value ranges of specific columns in a tuple.
database	A collection of data that is stored together and can be accessed, managed, and updated. Data in a view in the database can be classified into the following types: numerals, full text, digits, and images.
DB instance	A database instance consists of a process in GaussDB(DWS) and files controlled by the process. GaussDB(DWS) installs multiple database instances on one physical node. GTM, CM, CN, and DN installed on cluster nodes are all database instances. A database instance is also called a logical node.
database HA	GaussDB(DWS) provides a highly reliable HA solution. Every logical node in GaussDB(DWS) is identified as a primary or standby node. Only one GaussDB(DWS) node is identified as primary at a time. When the HA system is deployed for the first time, the primary server synchronizes all data from each standby server (full synchronization). The HA system then synchronizes only data that is new or has been modified from each standby server (incremental synchronization). When the HA system is running, the primary server can receive data read and write operation requests and the standby servers only synchronize logs.
database file	A binary file that stores user data and the data inside the database system.
data flow operator	An operator that exchanges data among query fragments. By their input/output relationships, data flows can be categorized into Gather flows, Broadcast flows, and Redistribution flows. Gather combines multiple query fragments of data into one. Broadcast forwards the data of one query fragment to multiple query fragments. Redistribution reorganizes the data of multiple query fragments and then redistributes the reorganized data to multiple query fragments.

Term	Description
data dictionary	A reserved table within a database which is used to store information about the database itself. The information includes database design information, stored procedure information, user rights, user statistics, database process information, database increase statistics, and database performance statistics.
deadlock	Unresolved contention for the use of resources.
index	An ordered data structure in the database management system. An index accelerates querying and the updating of data in database tables.
statistics	Information that is automatically collected by databases, including table-level information (number of tuples and number of pages) and column-level information (column value range distribution histogram). Statistics in databases are used to estimate the cost of execution plans to find the plan with the lowest cost.
projection	Projection is a unary operation. It uses specified attribute values from tables to form a new table and is represented by $\Pi_{\mathbf{A}}(\mathbf{R})$, where $\mathbf{A}$ indicates the table containing attribute names (column names) and $\mathbf{R}$ indicates the table name.
stop word	In computing, stop words are words which are filtered out before or after processing of natural language data (text), saving storage space and improving search efficiency.
U - Z	
vacuum	A thread that is periodically started up by a database to clear junk tuples. Multiple Vacuum threads can be started concurrently by setting a parameter.
verbose	The VERBOSE option specifies the information to be displayed.
WAL	Write-ahead logging (WAL) is a standard method for logging a transaction. Corresponding logs must be written into a permanent device before a data file (carrier for a table and index) is modified.
WAL Receiver	A thread created by the standby server during database duplication. The thread is used to receive data and commands from the primary server and to tell the primary server that the data and commands have been acknowledged. Only one WAL receiver thread can run on one standby server.
WAL Sender	A thread created on the primary server when the primary server has received a connection request from a standby server during database replication. This thread is used to send data and commands to standby servers and to receive responses from the standby servers. Multiple WAL Sender threads may run on one primary server. Each WAL Sender thread corresponds to a connection request initiated by a standby server.

Term	Description
WAL Writer	A thread for writing redo logs that are created when a database is started. This thread is used to write logs in the memory to a permanent device, such as a disk.
WLM	The WorkLoad Manager (WLM) is a module for controlling and allocating system resources in GaussDB(DWS).
Xlog	A transaction log. A logical node can have only one Xlog file.
xDR	X detailed record. It refers to detailed records on the user and signaling plans and can be categorized into charging data records (CDRs), user flow data records (UFDRs), transaction detail records (TDRs), and data records (SDRs).
network backup	Network backup provides a comprehensive and flexible data protection solution to Microsoft Windows, UNIX, and Linux platforms. Network backup can back up, archive, and restore files, folders, directories, volumes, and partitions on a computer.
physical node	A physical machine or device.
system catalog	A table storing meta information about the database. The meta information includes user tables, indexes, columns, functions, and the data types in a database.
selection	Selection is a unary operation on a table. It uses rows selected from a table based on given conditions to form a new table and is represented by $\sigma F(R)$, where σ is the selection operator, F indicates a conditional expression, and R indicates a table.
compression	Data compression, source coding, or bit-rate reduction involves encoding information that uses fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying and removing unnecessary or unimportant information. The process of reducing the size of a data file is commonly referred as data compression, although its formal name is source coding (coding done at the source of the data, before it is stored or transmitted).
consistency	One of the ACID features of database transactions. Consistency is a database status. In such a status, data in the database must comply with integrity constraints.
metadata	Data that provides information about other data. Metadata describes the source, size, format, or other characteristics of data. In database columns, metadata explains the content of a data warehouse.

Term	Description
atomicity	One of the ACID features of database transactions. Atomicity means that a transaction is composed of an indivisible unit of work. All operations performed in a transaction must either be committed or uncommitted. If an error occurs during transaction execution, the transaction is rolled back to the state when it was not committed.
dirty page	A page that has been modified and is not written to a permanent device.
incremental backup	Incremental backup stores all files changed since the last valid backup.
incremental synchronization	A data synchronization mechanism in the database HA solution. Only data modified since the last synchronization is synchronized to the standby server.
primary server	A node that receives data read and write operations in the GaussDB(DWS) HA system and works with all standby servers. At any time, only one node in the HA system is identified as the primary server.
thesaurus	Standardized words or phrases that express document themes and are used for indexing and retrieval.
dump file	A specific type of the trace file. A dump is typically a one-time output of diagnostic data in response to an event, whereas a trace tends to be continuous output of diagnostic data.
resource pool	Resource pools used for allocating resources in GaussDB(DWS). By binding a user to a resource pool, you can limit the priority of the jobs executed by the user and resources available to the jobs.
minimum restoration point	A method used by GaussDB(DWS) to ensure data consistency. During startup, GaussDB(DWS) checks consistency between the latest WAL logs and the minimum restoration point. If the record location of the minimum restoration point is greater than that of the latest WAL logs, the database fails to start.

A Change History

Release Date	Description
2020-12-30	This issue is the twenty-first official release.
2020-11-10	This issue is the twentieth official release. The following sections are deleted: • SQL Reference (An independent document <i>Data Warehouse Service SQL Syntax Reference</i> is provided.) • Full Text Search
2020-08-24	This issue is the nineteenth official release. The following sections are added: • Using gs_dump and gs_dumpall to Export Data • gs_dump • gs_dumpall The sections related to best practices are deleted (An independent document <i>Data Warehouse Service Best Practices</i> is provided).
2020-07-31	This issue is the eighteenth official release.
2020-07-10	This issue is the seventeenth official release. The following section is added: • Syntax Compatibility Differences Between Teradata and Oracle The following sections are modified: • Tutorial: Development Using JDBC or ODBC • Optimizing Statistics

Release Date	Description
2020-04-20	This issue is the sixteenth official release. The following sections are added: • Oracle compatibility - synonym – PG_SYNONYM – ALL_SYNONYMS – DBA_SYNONYMS – USER_SYNONYMS • KPI monitoring - SQL adaptation – GLOBAL_WORKLOAD_SQL_COUNT – GLOBAL_WORKLOAD_SQL_ELAPSE_TIME – GS_INSTR_UNIQUE_SQL – PGXC_WORKLOAD_SQL_COUNT – PGXC_WORKLOAD_SQL_ELAPSE_TIME – PGXC_INSTR_UNIQUE_SQL – GS_WORKLOAD_SQL_COUNT – GS_WORKLOAD_SQL_ELAPSE_TIME – PV_INSTANCE_TIME – PV_SESSION_TIME • Transaction enhancement – PGXC_VARIABLE_INFO – PGXC_GTM_SNAPSHOT_STATUS • plan table – PLAN_TABLE – PLAN_TABLE_DATA • Row- and column-level data access control (Row-Level Access Control) • Quick detection of storage skew – PG_STAT_ALL_TABLES – Query and Index Statistics Collector
2019-04-30	This issue is the fifteenth official release. The following sections are modified: • Express Functions

Release Date	Description
2019-04-10	This issue is the fourteenth official release. The following sections are added: ● ANALYZE enhancement: Doing VACUUM to a Table ● Optimization of data skew calculation: Skew Hints ● Read a page from the standby node if it is damaged on the primary node. – PGXC_STAT_BAD_BLOCK – PG_STAT_BAD_BLOCK ● Enhancement of logical cluster features: GS_TOTAL_NODEGROUP_MEMORY_DETAIL ● Enhancement of distributed execution frameworks: PGXC_STAT_ACTIVITY ● Transaction Isolation ● Record type supported: record ● GOTO syntax: GOTO Statements ● System catalog and view adaptation: – ALL_CONSTRAINTS – ALL_CONS_COLUMNS – ALL_IND_COLUMNS – ALL_INDEXES – ALL_TAB_COMMENTS – DBA_COL_COMMENTS – DBA_CONSTRAINTS – DBA_CONS_COLUMNS – DBA_IND_COLUMNS – DBA_IND_EXPRESSIONS – DBA_IND_PARTITIONS – DBA_TAB_COMMENTS – USER_COL_COMMENTS – USER_CONSTRAINTS – USER_CONS_COLUMNS – USER_IND_COLUMNS – USER_IND_EXPRESSIONS – USER_TAB_COMMENTS The following sections are modified: ● GDS enhancement: – Installing, Configuring, and Starting GDS – Creating a GDS Foreign Table ● Enhancement of resource management and concurrency control: I/O Priority Scheduling

Release Date	Description
	• Enhancement of distributed execution frameworks: – SQL Execution Plan – GS_WLM_SESSION_HISTORY – GS_WLM_SESSION_INFO – PG_COMM_RECV_STREAM – PG_COMM_SEND_STREAM – PGXC_COMM_RECV_STREAM – PGXC_COMM_SEND_STREAM – PGXC_GROUP • Optimization of the column-oriented hash agg algorithm: SQL Execution Plan Details • Optimization of data skew calculation: Optimizing Data Skew • Enhancement of key operators and function pushdown: Optimizing Statement Pushdown • Enhancement of logical cluster features: – PG_AUTHID – PG_RESOURCE_POOL – PGXC_GET_TABLE_SKEWNESS • 64-bit transaction IDs: – PG_CLASS – PG_DATABASE – PG_PARTITION – PG_PROC • System catalog and view adaptation: – ALL_DEPENDENCIES – ALL_TAB_COLUMNS – ALL_VIEWS – DBA_IND_PARTITIONS – DBA_INDEXES – DBA_TAB_COLUMNS – DBA_TAB_PARTITIONS – DBA_TABLES – USER_INDEXES – USER_TAB_COLUMNS – USER_TAB_PARTITIONS – USER_TABLES • Adding of the GUC parameter support_batch_bind: Developer Operations

Release Date	Description
	- Modification of the GUC parameter behavior_compat_options: Miscellaneous Parameters - Advanced package supported: DBMS_LOB
2019-02-28	This issue is the thirteenth official release. The following section is modified: Workload Management The following sections are added: Real-Time TopSQL Historical TopSQL Example for Querying for Top SQLsExample for Querying for Top SQLs
2019-02-01	This issue is the twelfth official release. The following sections are modified: Import Modes Querying User Resources The following sections are added: HA Transaction Processing High Query Performance Importing Stream Data from DIS to GaussDB(DWS) Using CDM to Migrate Data to GaussDB(DWS) Using the DSC Tool to Migrate SQL Scripts PG_TOTAL_USER_RESOURCE_INFO
2018-11-24	This issue is the eleventh official release. The following sections are modified: PL/Java User-defined Functions Querying User Resources The following sections are added: PG_TOTAL_USER_RESOURCE_INFO
2018-10-30	This issue is the tenth official release. The following sections are added: PG_STATISTIC_EXT The following section is modified: PL/Java User-defined Functions

Release Date	Description
2018-09-30	This issue is the ninth official release. The following sections are added: • DWS supports PostGIS function. – PostGIS – Using PostGIS – PostGIS Support and Constraints – OPEN SOURCE SOFTWARE NOTICE (For PostGIS) • JavaUDF interface: PL/Java User-defined Functions • Optimizing SQL Self-Diagnosis • The following sections about plan hints: – Plan Hint Optimization – Join Order Hints – Join Operation Hints – Rows Hints – Stream Operation Hints – Scan Operation Hints – Sublink Name Hints – Hint Errors, Conflicts, and Other Warnings – Plan Hint Cases • Node QPS view: GS_SQL_COUNT • Multi-column statistics view: PG_EXT_STATS • Defragmentation view: PGXC_GET_STAT_ALL_TABLES • Node QPS view: PGXC_SQL_COUNT The following sections are modified: • GUC parameter track_sql_count: Query and Index Statistics Collector • Description about multi-column statistics: Updating Statistics
2018-08-30	This is the eighth official release. The following sections are added: • OBS Query Overview • Querying Data on MRS The following sections are modified: • About Parallel Data Import from OBS

Release Date	Description
2018-07-10	This issue is the seventh official release. The following sections are added: • Using a gsql Meta-Command to Import Data • Data Import Using COPY FROM STDIN The following sections are modified: • Technical Specifications
2018-05-15	This issue is the sixth official release, The following sections are added: • Development and Design Proposal • Importing Data In Parallel Using GDS • Exporting Data In Parallel Using GDS • Viewing Parameter Values • Resource Load Management Overview • Information Schema
2018-04-09	This is the fifth official release. The following sections are added: • Target Readers The following sections are modified: • Technical Specifications • 8.4-Creating a User and Granting Permissions • Concurrently Importing Data from OBS • Importing Data From MRS to a Data Warehouse Cluster • Parallel OBS Data Export • OBS Query Overview
2018-03-02	This issue is the fourth official release. The following sections are added: • Running the INSERT Statement to Insert Data • Low Query Efficiency The following sections are modified: • About Parallel Data Import from OBS The following sections are deleted: • SQL error code reference: The document has been released as a separate document.

Release Date	Description
2018-02-02	This issue is the third official release. The following sections are deleted: • Product Positioning • Product Features • JDBC Application Developer Guide • ODBC Application Developer Guide
2017-12-15	This issue is the second official release.
2017-11-24	This issue is the first official release.