GaussDB

Developer Guide(Distributed_V2.0-10.x)

Issue	01
Date	2026-09-16

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2026. All rights reserved.

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

Trademarks and Permissions

and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Database System Overview 1
1.1 Database Logical Architecture 1
1.2 Query Request Handling Process 2
1.3 Managing Transactions 3
2 Database Security 6
2.1 Users and Permissions 6
2.1.1 Default Permission Mechanism 6
2.1.2 Administrators 7
2.1.3 Users 9
2.1.4 Roles 10
2.1.5 Schemas 12
2.1.6 User Permissions 13
2.1.7 Row-Level Security Policy 14
2.2 Database Audit 16
3 Database Quick Start 19
3.1 Connecting to a Database 19
3.1.1 Using gsql to Connect to a Database 19
3.1.2 APIs 21
3.2 Operating on a Database 22
3.2.1 Creating a Database User 22
3.2.2 Managing Databases in Non-Multi-Primary Mode 23
3.2.3 Managing Tablespaces 26
3.2.4 Managing Tables 29
3.2.4.1 Creating a Table 29
3.2.4.2 Inserting Data into Tables 30
3.2.4.3 Updating Data in a Table 34
3.2.4.4 Viewing Data 36
3.2.4.5 Deleting Data from a Table 36
3.2.5 Querying System Catalogs 37
3.2.6 Other Operations 39
3.2.6.1 Managing Schemas 39
3.2.6.2 Managing Partitioned Tables 42

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.6.3 Managing Indexes....47
3.2.6.4 Managing Views....51
3.2.6.5 Managing Sequences....52
3.2.6.6 Managing Scheduled Jobs....54
3.3 Managing Permissions....58
3.4 Using Stored Procedures....60
4 Development and Design Proposal....65
4.1 Overview....65
4.2 Database Design Specifications....66
4.2.1 General Specifications....66
4.2.2 Deployment Specifications....68
4.2.3 Database Object Naming Specifications....69
4.2.4 Database and Schema Design Specifications....71
4.2.5 Permission Design Specifications....73
4.2.6 Character Set Design Specifications....73
4.2.7 Table Design Specifications....74
4.2.8 Column Design Specifications....79
4.2.9 Index Design Specifications....83
4.2.10 Function/Stored Procedure Design Specifications....84
4.2.11 Constraint Design....84
4.2.12 View and Joined Table Design....85
4.3 Database Programming Specifications....85
4.3.1 GUC Parameter Programming Specifications....85
4.3.2 Object Access Programming Specifications....85
4.3.3 WHERE....86
4.3.4 SELECT....87
4.3.5 INSERT....89
4.3.6 UPDATE....89
4.3.7 DELETE....89
4.3.8 Join Query....90
4.3.9 Subquery....91
4.3.10 Transaction....91
4.3.11 SQL Compilation....92
4.4 Client Programming Specifications....96
4.4.1 JDBC....96
4.4.2 Connection Control Specifications....99
4.4.3 C/C++ Application Compilation Specifications....99
5 Application Development Guide....105
5.1 GaussDB Application Development Guide....105
5.2 Development Specifications....117
5.3 Development Based on JDBC....118
5.3.1 Development Process....118

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.2 Development Procedure 119
5.3.2.1 Obtaining the Driver JAR Package and Configuring the JDK Environment 119
5.3.2.2 Connecting to a Database 124
5.3.2.2.1 Connection Methods 124
5.3.2.2.2 Connection in Non-encrypted Mode 129
5.3.2.2.3 Connection in SSL Mode 131
5.3.2.2.4 Connecting to a Database in UDS Mode 132
5.3.2.2.5 Connection Parameter Reference 133
5.3.2.3 Executing SQL Statements 181
5.3.2.4 Processing the Result Set 187
5.3.2.5 Closing the Database Connection 190
5.3.3 Typical Application Development Examples 190
5.3.3.1 Configuring Connection Parameters in Different Scenarios 190
5.3.3.2 Handling SQL Exceptions 195
5.3.3.3 Creating and Calling a Stored Procedure 196
5.3.3.4 Obtaining the Return Value of a Function 198
5.3.3.5 Batch Query 199
5.3.3.6 Using the Streaming Read Function 201
5.3.3.7 SQL Retry at an Application Layer 204
5.3.3.8 Importing and Exporting Data Through Local Files 207
5.3.3.9 Logical Replication (Non-Multi-Primary) 209
5.3.3.10 Monitoring Database Connections 219
5.3.3.11 Using the Array Type 221
5.3.3.12 Using the Struct Type 232
5.3.4 JDBC API Reference 241
5.3.4.1 Common Packages and APIs 241
5.3.4.2 java.sql.Connection 243
5.3.4.3 java.sql.CallableStatement 252
5.3.4.4 java.sql.DatabaseMetaData 259
5.3.4.5 java.sql.Driver 269
5.3.4.6 java.sql.PreparedStatement 269
5.3.4.7 java.sql.ResultSet 273
5.3.4.8 java.sql.ResultSetMetaData 282
5.3.4.9 java.sql.Statement 284
5.3.4.10 java.sql.Clob 286
5.3.4.11 java.sql.Array 288
5.3.4.12 java.sql.Struct 290
5.3.4.13 java.sql.SQLException 291
5.3.4.14 java.io.IOException 292
5.3.4.15 javax.sql.ConnectionPoolDataSource 293
5.3.4.16 javax.sql.DataSource 293
5.3.4.17 javax.sql.PooledConnection 293

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.18 javax.naming.Context 294
5.3.4.19 CopyManager 294
5.3.4.20 PGReplicationConnection 296
5.3.4.21 PGReplicationStream 298
5.3.4.22 ChainedStreamBuilder 299
5.3.4.23 ChainedCommonStreamBuilder 300
5.3.4.24 PGobject 301
5.3.4.25 LogSequenceNumber 302
5.4 Development Based on ODBC 303
5.4.1 Development Process 303
5.4.2 Development Procedure 303
5.4.2.1 Obtaining the Source Code Package, ODBC Package, and Dependency Library 303
5.4.2.2 Connecting to a Database 305
5.4.2.2.1 Configuring a Data Source on Linux 305
5.4.2.2.2 Configuring a Data Source on Windows 310
5.4.2.2.3 APIs for Connecting to a Database 314
5.4.2.3 Executing SQL Statements 316
5.4.2.4 Processing the Result Set 317
5.4.2.5 Closing the Connection 318
5.4.3 Typical Application Development Examples 318
5.4.3.1 Typical Application Scenarios and Configurations 318
5.4.3.2 Obtaining and Processing Data in a Database 322
5.4.3.3 Batch Binding 324
5.4.3.4 High-Performance Binding Type 329
5.4.3.5 Enabling a Connection Pool 337
5.4.4 ODBC API Reference 339
5.4.4.1 SQLAllocEnv 339
5.4.4.2 SQLAllocConnect 339
5.4.4.3 SQLAllocHandle 340
5.4.4.4 SQLAllocStmt 341
5.4.4.5 SQLBindCol 341
5.4.4.6 SQLBindParameter 342
5.4.4.7 SQLColAttribute 344
5.4.4.8 SQLConnect 346
5.4.4.9 SQLDisconnect 347
5.4.4.10 SQLExecDirect 348
5.4.4.11 SQLExecute 349
5.4.4.12 SQLFetch 350
5.4.4.13 SQLFreeStmt 350
5.4.4.14 SQLFreeConnect 351
5.4.4.15 SQLFreeHandle 351
5.4.4.16 SQLFreeEnv 352

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.17 SQLPrepare 352
5.4.4.18 SQLGetData 353
5.4.4.19 SQLGetDiagRec 354
5.4.4.20 SQLSetConnectAttr 357
5.4.4.21 SQLSetEnvAttr 358
5.4.4.22 SQLSetStmtAttr 359
5.5 Development Based on libpq 360
5.5.1 Development Process 360
5.5.2 Development Procedure 360
5.5.2.1 Obtaining the Release Package, Dependency Libraries, and Header Files 361
5.5.2.2 Connecting to a Database 362
5.5.2.3 Executing SQL Statements 364
5.5.2.4 Processing the Result Set 365
5.5.2.5 Closing the Connection 365
5.5.3 Typical Application Development Examples 366
5.5.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results 366
5.5.3.2 Executing Prepared Statements 368
5.5.3.3 Binding Parameters and Returning a Binary Result 370
5.5.4 libpq API Reference 373
5.5.4.1 Overview 373
5.5.4.2 Database Connection Control Functions 374
5.5.4.2.1 PQconnectdbParams 374
5.5.4.2.2 PQconnectdb 375
5.5.4.2.3 PQconninfoParse 376
5.5.4.2.4 PQconnectStart 376
5.5.4.2.5 PQerrorMessage 377
5.5.4.2.6 PQsetdbLogin 377
5.5.4.2.7 PQfinish 378
5.5.4.2.8 PQreset 379
5.5.4.2.9 PQstatus 379
5.5.4.3 Database Statement Execution Functions 380
5.5.4.3.1 PQexec 380
5.5.4.3.2 PQprepare 382
5.5.4.3.3 PQclear 383
5.5.4.3.4 PQfnumberEx 384
5.5.4.3.5 PQexecParams 384
5.5.4.3.6 PQexecParamsBatch 385
5.5.4.3.7 PQexecPrepared 386
5.5.4.3.8 PQexecPreparedBatch 387
5.5.4.4 Asynchronous Command Processing Functions 388
5.5.4.4.1 PQsendQuery 388
5.5.4.4.2 PQsendQueryParams 389

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.4.3 PQsendPrepare 390
5.5.4.4.4 PQsendQueryPrepared 391
5.5.4.4.5 PQflush 392
5.5.4.5 Query Cancellation Functions 393
5.5.4.5.1 PQgetCancel 393
5.5.4.5.2 PQfreeCancel 394
5.5.4.5.3 PQcancel 394
5.5.4.6 Functions for Processing Database Result 395
5.5.4.6.1 PQgetvalue 395
5.5.4.6.2 PQnfields 395
5.5.4.6.3 PQntuples 396
5.5.4.6.4 PQfname 397
5.5.4.6.5 PQresultStatus 397
5.6 Development Based on Psycopg 398
5.6.1 Development Process 398
5.6.2 Development Procedure 399
5.6.2.1 Preparing an Environment 399
5.6.2.2 Connecting to a Database 405
5.6.2.2.1 Connection Methods 405
5.6.2.2.2 Connection in Non-encrypted Mode 405
5.6.2.2.3 Connection in SSL Mode 406
5.6.2.2.4 Connection Parameter Reference 408
5.6.2.3 Executing SQL Statements 414
5.6.2.4 Processing the Result Set 415
5.6.2.5 Closing the Database Connection 415
5.6.3 Typical Application Development Examples 416
5.6.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results 416
5.6.3.2 Using Logs 417
5.6.4 Psycopg API Reference 418
5.6.4.1 psycopg2.connect() 418
5.6.4.2 connection.cursor() 419
5.6.4.3 cursor.execute(query,vars_list) 419
5.6.4.4 cursor.executemany(query,vars_list) 420
5.6.4.5 connection.commit() 421
5.6.4.6 connection.rollback() 421
5.6.4.7 cursor.fetchone() 422
5.6.4.8 cursor.fetchmany(size) 422
5.6.4.9 cursor.fetchall() 423
5.6.4.10 cursor.close() 424
5.6.4.11 connection.close() 425
5.7 Development Based on the Go Driver 425
5.7.1 Development Process 425

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.7.2 Development Procedure 426
5.7.2.1 Preparing an Environment 426
5.7.2.2 Connecting to a Database 429
5.7.2.2.1 Connection in Non-encrypted Mode 429
5.7.2.2.2 Connection in SSL Mode 433
5.7.2.2.3 Connection Parameter Reference 435
5.7.2.3 Executing SQL Statements 442
5.7.2.4 Processing the Result Set 443
5.7.2.5 Closing the Database Connection 444
5.7.3 Typical Application Development Examples 444
5.7.4 Go APIs 448
5.7.4.1 type DB 448
5.7.4.2 type Stmt 451
5.7.4.3 type Tx 452
5.7.4.4 type Rows 455
5.7.4.5 type Row 456
5.7.4.6 type ColumnType 456
5.7.4.7 type Result 457
5.8 Development Based on ECPG 457
5.8.1 Development Process 457
5.8.2 Preparing Components for ECPG-based Development 459
5.8.3 Embedded SQL-C Program Development 460
5.8.3.1 Foundational Concepts 460
5.8.3.2 Declaring Host Variables 462
5.8.3.3 Managing Database Connections 463
5.8.3.4 Executing SQL Statements 465
5.8.3.5 Processing the Result Set 467
5.8.3.6 Closing a Connection 470
5.8.3.7 Embedded SQL Syntax Reference 471
5.8.3.7.1 ALLOCATE DESCRIPTOR 471
5.8.3.7.2 CONNECT 471
5.8.3.7.3 DEALLOCATE DESCRIPTOR 474
5.8.3.7.4 DECLARE 475
5.8.3.7.5 DESCRIBE 476
5.8.3.7.6 DISCONNECT 476
5.8.3.7.7 EXECUTE IMMEDIATE 477
5.8.3.7.8 GET DESCRIPTOR 478
5.8.3.7.9 OPEN 479
5.8.3.7.10 PREPARE 480
5.8.3.7.11 SET AUTOCOMMIT 481
5.8.3.7.12 SET CONNECTION 481
5.8.3.7.13 SET DESCRIPTOR 482

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.7.14 TYPE 482
5.8.3.7.15 VAR 484
5.8.3.7.16 WHENEVER 484
5.8.4 ECPG-based Preprocessing, Compilation, and Execution 486
5.8.5 Typical Application Development Examples 489
5.8.5.1 Creating a Table, Inserting Data, and Querying Data 489
5.8.5.2 Date, Timestamp, and Interval Parsing 492
5.8.5.3 Parsing and Calculating Numeric Types 495
5.8.6 ECPG API Reference 501
5.8.6.1 Overview 501
5.8.6.2 Interval Type 501
5.8.6.3 Numeric Type 502
5.8.6.4 Date Type 506
5.8.6.5 Timestamp Type 510
5.9 Compatibility Reference 513
5.10 Appendix 514
5.10.1 JDBC 514
5.10.1.1 Data Type Mapping 514
5.10.1.2 Data Type Conversion Rules 515
5.10.1.3 Log Management 587
5.10.1.4 Troubleshooting 591
5.10.1.4.1 Incorrect batchMode Settings 591
5.10.1.4.2 Error Is Reported When Verification Is Enabled for Data Insertion in the Hibernate Framework 592
5.10.1.4.3 Error Is Reported or Connection Is Blocked in SSL Mode 594
5.10.2 ODBC 595
5.10.2.1 Configuration Parameters 595
5.10.3 libpq 605
5.10.3.1 Connection Parameters 605
5.10.4 ECPG 615
5.10.4.1 Prepared Statements 615
5.10.4.2 Data Types and Host Variables 615
5.10.4.2.1 Overview 616
5.10.4.2.2 DECLARE Section 616
5.10.4.2.3 Retrieving Query Results 617
5.10.4.2.4 Data Type Mapping 617
5.10.4.2.5 Handling Character Strings 618
5.10.4.2.6 Host Variables with Non-primitive Types 619
5.10.4.2.7 Accessing Special Data Types 622
5.10.4.2.8 Handling Non-primitive SQL Data Types 623
5.10.4.3 Executing Dynamic SQL Statements 626
5.10.4.3.1 Overview 626
5.10.4.3.2 Executing a Statement Without a Result Set 626

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.10.4.3.3 Executing a Statement with Input Parameters....627
5.10.4.3.4 Executing a Statement with a Result Set....627
5.10.4.4 Error Handling....628
5.10.4.4.1 Overview....628
5.10.4.4.2 Setting Callbacks....628
5.10.4.4.3 sqlca....629
5.10.4.4.4 SQLSTATE and SQLCODE....630
5.10.4.5 Using Library Functions....634
5.10.4.6 SQL Descriptor Areas....637
5.10.4.6.1 Overview....637
5.10.4.6.2 Named SQL Descriptor Areas....637
5.10.4.6.3 SQLDA....639
5.10.4.7 Compatibility Between ECPG and Pro*C....642
5.10.5 Parameters Related to Log Output....643
6 User-Defined Functions....648
6.1 PL/SQL Functions....648
7 Feature Usage Guidelines....650
7.1 Autonomous Transaction....650
7.1.1 Specifications and Limitations....650
7.1.2 Autonomous Transactions Supported in Stored Procedures....656
7.1.3 Autonomous Transactions Supported in Anonymous Blocks....657
7.1.4 Autonomous Transactions Supported in Functions....657
7.1.5 Package Supporting Autonomous Transaction....658
7.2 Materialized View....660
7.2.1 Complete-Refresh Materialized View....660
7.2.1.1 Overview....660
7.2.1.2 Support and Limitations....660
7.2.1.3 Usage....661
7.2.2 Fast-Refresh Materialized View....662
7.2.2.1 Overview....662
7.2.2.2 Support and Limitations....662
7.2.2.3 Usage....663
7.3 Encrypted Equality Query....664
7.3.1 Overview....664
7.3.2 Using gsql to Operate on an Encrypted Database....668
7.3.3 Using JDBC to Operate on an Encrypted Database....671
7.3.4 Backward Compatibility and Security Enhancement....677
7.3.5 Encrypted Functions and Stored Procedures....680
7.4 Transparent Data Encryption....682
7.5 Ledger Database....689
7.5.1 Overview....689
7.5.2 Viewing Historical Operation Records in the Ledger....692

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.5.3 Verifying Ledger Data Consistency 694
7.5.4 Archiving a Ledger Database 695
7.5.5 Repairing a Ledger Database 696
7.6 Label-based Mandatory Access Control 699
7.6.1 Defining Security Labels 699
7.6.2 Applying Security Labels 700
7.6.3 Label-based Mandatory Access Control Checking 701
7.7 Logical Replication 702
7.7.1 Logical Decoding (Non-Multi-Primary) 702
7.7.1.1 Overview 702
7.7.1.2 Logical Decoding Options 713
7.7.1.3 Logical Decoding by SQL Functions 729
7.7.1.4 Logical Data Replication Using Streaming Decoding 730
7.7.1.5 Logical Decoding of DDL Statements 730
7.7.1.6 Data Retrieval by Logical Decoding 741
7.8 Partitioned Table 742
7.8.1 Introduction to Table Partitioning 742
7.8.1.1 Background 742
7.8.1.2 Table Partitioning Technology 742
7.8.1.3 Optimized Data Queries by Partition 743
7.8.1.4 Partition O&M 744
7.8.2 Introduction to Partitioned Tables 744
7.8.2.1 Basic Concepts 745
7.8.2.1.1 Partitioned Table 745
7.8.2.1.2 Partition 746
7.8.2.1.3 Partition Key 747
7.8.2.2 Partitioning Policies 747
7.8.2.2.1 Range Partitioning 748
7.8.2.2.2 Hash Partitioning 750
7.8.2.2.3 List Partitioning 751
7.8.2.2.4 Subpartitioning 752
7.8.2.2.5 Impact on Import Performance 755
7.8.2.3 Basic Usage of Partitions 757
7.8.2.3.1 Creating Partitioned Tables 757
7.8.2.3.2 Using and Managing Partitioned Tables 759
7.8.2.3.3 DQL/DML on a Partitioned Table 760
7.8.3 Query Optimizations on Partitioned Tables 762
7.8.3.1 Partition Pruning 762
7.8.3.1.1 Static Pruning for Partitioned Tables 763
7.8.3.1.2 Dynamic Pruning for Partitioned Tables 767
7.8.3.2 Optimized Execution of Partition Operators 774
7.8.3.2.1 Partition Iterator Elimination 774

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.3.2.2 Merge Append....775
7.8.3.2.3 Max/Min....777
7.8.3.2.4 Dynamic Pruning....779
7.8.3.3 Partitioned Index....780
7.8.3.4 Partitioned Table Statistics....783
7.8.3.4.1 Collecting Statistics in Cascading Mode....783
7.8.3.4.2 Collecting Partition-Level Statistics....786
7.8.3.5 Partition-wise Join....790
7.8.3.5.1 Partition-wise Join in Non-SMP Scenarios....790
7.8.3.5.2 Full Partition-wise Join in SMP Scenarios....792
7.8.3.5.3 Partial Partition-wise Join in SMP Scenarios....797
7.8.4 Partitioned Table O&M....800
7.8.4.1 ADD PARTITION....800
7.8.4.1.1 Adding a Partition to a Range Partitioned Table....801
7.8.4.1.2 Adding a Partition to a List Partitioned Table....801
7.8.4.1.3 Adding a Partition to a Subpartitioned Table....801
7.8.4.1.4 Adding a Subpartition to a Subpartitioned Table....802
7.8.4.2 DROP PARTITION....802
7.8.4.2.1 Dropping a Partition from a Partitioned Table....803
7.8.4.2.2 Dropping a Partition from a Subpartitioned Table....803
7.8.4.2.3 Dropping a Subpartition from a Subpartitioned Table....804
7.8.4.3 EXCHANGE PARTITION....804
7.8.4.3.1 Exchanging a Partition in a Partitioned Table....806
7.8.4.3.2 Exchanging a Subpartition in a Subpartitioned Table....806
7.8.4.4 TRUNCATE PARTITION....806
7.8.4.4.1 Clearing a Partition in a Partitioned Table....807
7.8.4.4.2 Clearing a Partition in a Subpartitioned Table....807
7.8.4.4.3 Clearing a Subpartition in a Subpartitioned Table....807
7.8.4.5 SPLIT PARTITION....808
7.8.4.5.1 Splitting a Partition in a Range Partitioned Table....808
7.8.4.5.2 Splitting a Partition in a List Partitioned Table....809
7.8.4.5.3 Splitting a Subpartition in a *-Range Subpartitioned Table....809
7.8.4.5.4 Splitting a Subpartition in a *-List Subpartitioned Table....810
7.8.4.6 MERGE PARTITION....811
7.8.4.6.1 Merging Partitions in a Partitioned Table....811
7.8.4.6.2 Merging Subpartitions in a Subpartitioned Table....811
7.8.4.7 MOVE PARTITION....812
7.8.4.7.1 Moving a Partition in a Partitioned Table....812
7.8.4.7.2 Moving a Subpartition in a Subpartitioned Table....812
7.8.4.8 RENAME PARTITION....812
7.8.4.8.1 Renaming a Partition in a Partitioned Table....812
7.8.4.8.2 Renaming a Partition in a Subpartitioned Table....812

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.4.8.3 Renaming a Subpartition in a Subpartitioned Table 813
7.8.4.8.4 Renaming the Index Partition for a Local Index 813
7.8.4.9 ROW MOVEMENT 813
7.8.4.10 UNUSABLE/REBUILD 814
7.8.4.10.1 Setting an Index as Unusable/Rebuilding an Index 814
7.8.4.10.2 Setting Local Index Partitions as Unusable/Rebuilding Local Index Partitions 814
7.8.5 Partition Concurrency Control 815
7.8.5.1 Standard Lock Design 815
7.8.5.2 DQL/DML-DQL/DML Concurrency 817
7.8.5.3 DQL/DML-DDL Concurrency 818
7.8.6 System Views & DFx Related to Partitioned Tables 821
7.8.6.1 System Views Related to Partitioned Tables 821
7.8.6.2 Built-in Tool Functions Related to Partitioned Tables 822
7.9 Storage Engine 825
7.9.1 Storage Engine Architecture 825
7.9.1.1 Overview 825
7.9.1.1.1 Static Compilation Architecture 825
7.9.1.1.2 Database Service Layer 826
7.9.1.2 Setting Up a Storage Engine 827
7.9.1.3 Storage Engine Update Description 828
7.9.1.3.1 GaussDB Kernel 505 828
7.9.1.3.2 GaussDB Kernel 503 828
7.9.1.3.3 GaussDB Kernel R2 829
7.9.2 Astore 829
7.9.2.1 Overview 829
7.9.3 Ustore 830
7.9.3.1 Overview 830
7.9.3.1.1 Ustore Features and Specifications 830
7.9.3.1.2 Examples 831
7.9.3.1.3 Best Practices of Ustore 832
7.9.3.2 Storage Format 836
7.9.3.2.1 RCR Uheap 836
7.9.3.2.2 UB-Tree 838
7.9.3.2.3 Undo 843
7.9.3.2.4 Enhanced TOAST 844
7.9.3.3 Ustore Transaction Model 849
7.9.3.3.1 Transaction Commit 849
7.9.3.3.2 Transaction Rollback 850
7.9.3.4 Flashback 850
7.9.3.4.1 Flashback Query 851
7.9.3.4.2 Flashback Table 853
7.9.3.4.3 Flashback DROP/TRUNCATE 855

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.9.3.5 Common View Tools....863
7.9.3.6 Common Problems and Troubleshooting Methods....867
7.9.3.6.1 Snapshot Too Old....867
7.9.3.6.2 Storage Test Error....868
7.9.3.6.3 Read-on-Standby Error: "Index scan conflicts with replay. Try again later. The original maxoff is <number>, but now it is <number>."....869
7.9.3.6.4 Write Performance Deteriorates Occasionally When a Large Number of Concurrent Updates Are Performed During Long Query Execution....870
7.9.4 Data Lifecycle Management: OLTP Table Compression....871
7.9.4.1 Introduction....871
7.9.4.2 Limitations....871
7.9.4.3 Specifications....875
7.9.4.4 Usage Guide....879
7.9.4.5 Setting the Maintenance Window Parameters....885
7.9.4.6 O&M Tips....887
7.10 Memory Management....895
7.11 Multi-thread Architecture....901
7.12 ABO....905
7.12.1 Cardinality Estimation....905
7.12.1.1 Overview....906
7.12.1.2 Technical Principles....907
7.12.1.3 Operation Guide....909
7.12.1.4 Best Practices....912
7.12.1.5 Typical Issues....913
7.12.2 Adaptive Cost Estimation....913
7.12.2.1 Overview....913
7.12.2.2 Technical Principle....916
7.12.2.3 Operation Guide....924
7.12.2.4 Typical Issues....929
7.13 CBO....929
7.13.1 Overview....929
7.13.2 Technical Principle....930
7.13.2.1 Statistics....930
7.13.2.1.1 Single-Column Statistics....931
7.13.2.1.2 Multi-Column Statistics....937
7.13.2.2 Path Generation....945
7.13.2.2.1 Scan Path Generation....946
7.13.2.2.2 Connection Path Generation....951
7.13.2.3 Plan Generation....955
7.13.2.3.1 Cost Estimation....956
7.13.2.3.2 Plan Selection....959
7.13.3 Operation Guide....963
7.13.4 Typical Issues....965

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.14 AutoHint for Intelligent Hint Recommendation....969
7.15 DSL Rule Engine....972
7.15.1 Overall Process of the DSL Rule Engine....972
7.15.2 DSL Rule Writing Specifications....994
7.15.2.1 DSL Rule Syntax....994
7.15.2.2 DSL Rule Examples and Description....1058
7.15.3 Best Practices for the DSL Rule Engine....1062
7.15.4 Troubleshooting....1064
7.16 Foreign Data Wrapper....1065
7.16.1 file_fdw....1065
7.17 GTM....1067
7.17.1 Overview....1067
7.17.2 GTM Modes....1069
7.17.2.1 GTM....1069
7.17.2.1.1 GTM Mode....1069
7.17.2.1.2 Technical Principles....1070
7.17.2.2 GTM-Lite....1070
7.17.2.2.1 GTM-Lite Mode....1070
7.17.2.2.2 Technical Principle....1071
7.17.2.3 GTM-Free....1071
7.17.2.3.1 GTM-Free Mode....1071
7.17.2.3.2 Technical Principles....1073
7.17.3 Operation Guide....1073
7.17.4 Best Practices for Distributed Mode....1073
7.17.5 Typical Issues....1074
7.18 Sensitive Data Discovery....1074
7.19 Dynamic Data Masking....1075
7.20 Hybrid Transactional/Analytical Processing....1077
7.20.1 Overview....1077
7.20.2 Usage Guide....1078
7.20.2.1 Parameter Settings....1078
7.20.2.2 Memory Control....1079
7.20.2.3 DDL....1081
7.20.2.3.1 Creating and Unloading an IMCV....1081
7.20.2.3.2 Other DDL Statements....1090
7.20.2.4 DML/DQL....1093
7.20.2.4.1 DML....1093
7.20.2.4.2 DQL....1094
7.20.2.4.3 Column-to-Row Hybrid Scan....1095
7.20.2.5 Transparent Routing for HTAP....1096
7.20.2.6 Column-Store Data Maintenance....1101
7.20.2.7 Standby Node Capabilities....1105

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.20.2.8 Column-Store Data Disk Extension 1105
7.20.2.9 Column-Store Data Compression 1106
7.21 Online DDL Operations 1107
7.22 Bucket Distributed Tables 1110
7.22.1 Hash Bucket tables 1111
7.22.1.1 Creating a Hash Bucket Table 1111
7.22.1.2 Bucket Indexes 1112
7.22.1.2.1 Recommended Application Scenarios of Bucket Indexes 1112
7.22.1.3 Feature Specifications and Limitations of the Hash Bucket Table 1114
7.22.1.4 Pruning a Hash Bucket Table 1115
7.22.2 Best Practices for Bucket Distributed Tables 1117
7.22.2.1 Setting segment_buffers 1117
7.22.2.2 Space Preallocation for a Bucket Distributed Table 1118
7.23 Recompilation upon Invalidation 1121
7.23.1 Overview 1121
7.23.2 Technical Principle 1123
7.23.3 Operation Guide 1123
7.23.4 FAQ 1126
7.24 Locating Stored Procedure Performance Problems 1129
7.24.1 Locating Performance Problems During Stored Procedure Execution 1129
7.24.1.1 Querying pg_stat_activity to View the Statements That Are Being Executed in a Stored Procedure 1129
7.24.1.1.1 Overview 1129
7.24.1.1.2 Technical Principle 1130
7.24.1.1.3 Operation Guide 1131
7.24.1.2 Tracing Statements Executed by a Stored Procedure 1132
7.24.1.2.1 Overview 1132
7.24.1.2.2 Technical Principle 1133
7.24.1.2.3 Operation Guide 1134
7.24.2 Locating Performance Problems After Stored Procedure Execution 1135
7.24.2.1 Slow SQL Statements in a Stored Procedure 1135
7.24.2.1.1 Overview 1136
7.24.2.1.2 Technical Principle 1137
7.24.2.1.3 Operation Guide 1138
7.24.3 Typical Issues 1140
7.25 Database Connection Driver Guide 1141
7.25.1 JDBC 1141
7.25.1.1 Overview 1141
7.25.1.2 Technical Principle 1141
7.25.1.3 Operation Guide 1145
7.25.1.3.1 Development Process 1145
7.25.1.3.2 Obtaining the Driver JAR Package and Configuring the JDK Environment 1146
7.25.1.3.3 Creating a Database Connection 1149

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.25.1.3.4 Executing SQL Statements....1151
7.25.1.3.5 Processing the Result Set....1153
7.25.1.3.6 Closing a Database Connection....1153
7.25.1.3.7 Best Practices....1154
7.25.1.4 Typical Issues....1154
7.25.1.4.1 Incorrect batchMode Settings....1154
7.25.1.4.2 An Error Is Reported When Validation Is Enabled for Data Insertion in the Hibernate Framework....1155
7.25.1.4.3 An Error Is Reported or A Connection Is Blocked in SSL Mode....1157
7.25.2 ODBC....1157
7.25.2.1 Overview....1158
7.25.2.2 Technical Principle....1158
7.25.2.3 Operation Guide....1162
7.25.2.3.1 Development Process....1162
7.25.2.3.2 Obtaining the Source Code Package, ODBC Package, and Dependency Library....1163
7.25.2.3.3 Creating a Database Connection....1164
7.25.2.3.4 Executing SQL Statements....1181
7.25.2.3.5 Processing the Result Set....1181
7.25.2.3.6 Closing a Connection....1182
7.25.2.3.7 Best Practices....1182
7.25.2.4 Typical Issues....1182
7.26 Online Scale-out Based on Hash Bucket Tables....1184
7.26.1 Overview....1184
7.26.2 Technical Principles....1186
7.26.3 Operation Guide....1189
7.26.4 Typical Issues....1199
7.27 Read on Standby....1200
7.27.1 Overview....1200
7.27.2 Technical Principle....1202
7.27.2.1 Three Replay Modes of Read on Standby....1202
7.27.2.2 Read on Standby in Distributed GTM-Free Mode....1206
7.27.2.3 Read on Standby in Distributed GTM-Lite Mode....1209
7.27.2.4 DR Read in Distributed GTM-Lite Mode....1214
7.27.3 Operation Guide....1219
7.27.3.1 Selecting a Replay Mode Based on Your Environment Specifications....1219
7.27.3.2 Procedure....1223
7.27.3.3 Precautions....1226
7.27.4 FAQ....1226
7.28 Anti-overloads....1235
7.28.1 SQL Statement Concurrency Control....1235
7.28.1.1 Overview....1235
7.28.1.2 Technical Principle....1237
7.28.1.3 Operation Guide....1238

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.28.1.4 Typical Issues 1239
7.28.2 Memory Overload Protection 1240
7.28.2.1 Overview 1240
7.28.2.2 Technical Principle 1240
7.28.2.3 Operation Guide 1242
7.28.2.4 Typical Issues 1242
7.28.3 Thread Pool Overload Protection 1243
7.28.3.1 Overview 1243
7.28.3.2 Technical Principle 1244
7.28.3.3 Operation Guide 1245
7.28.3.4 Typical Issues 1246
7.28.4 Connection Thread Pool 1246
7.28.4.1 Overview 1246
7.28.4.2 Technical Principle 1247
7.28.4.3 Operation Guide 1248
7.28.5 Overload Escape in Concurrency Scenarios 1248
7.28.5.1 Overview 1249
7.28.5.2 Technical Principle 1249
7.28.5.3 Operation Guide 1250
7.28.5.4 Typical Issues 1251
7.29 Unix Domain Socket O&M Channel 1251
7.29.1 Overview 1251
7.29.2 Technical Principles 1252
7.29.3 Operation Guide 1266
7.29.4 FAQ 1267
7.30 CM Arbitration and Fault Detection Principles 1268
7.30.1 Overview 1268
7.30.2 Technical Principles 1269
7.30.2.1 Arbitration Principle 1269
7.30.2.1.1 DN Arbitration 1269
7.30.2.1.2 CN Arbitration 1270
7.30.2.1.3 GTM Arbitration 1271
7.30.2.1.4 CM Server Arbitration 1271
7.30.2.1.5 ETCD Arbitration 1272
7.30.2.2 Fault Detection Principle 1273
7.30.2.2.1 Disk Check 1273
7.30.2.2.2 Zombie DN Detection 1274
7.30.2.2.3 First Aid Capability 1274
7.30.2.2.4 Hardware Troubleshooting 1274
7.30.3 Operation Guide 1275
7.30.4 FAQ 1275
7.31 Principles of Intra-cluster and Inter-cluster Switchover 1275

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.31.1 Intra-cluster Switchover 1275
7.31.1.1 Overview 1276
7.31.1.2 Technical Principle 1276
7.31.1.3 Operation Guide 1277
7.31.1.4 Typical Issues 1278
7.31.2 Inter-cluster Switchover 1278
7.31.2.1 Overview 1278
7.31.2.2 Technical Principle 1282
7.31.2.3 Operation Guide 1283
7.31.2.3.1 Streaming Replication–based Remote DR Drill Practice Reference 1283
7.31.2.3.2 Intra-City Dual-DC HA Solution DR Drill Practice Reference 1289
7.31.2.4 Typical Issues 1294
7.31.2.4.1 Exceptions in Planned Switchover for Two-City Three-DC DR 1294
7.31.2.4.2 Exceptions in Planned Switchover of Intra-City Dual-DC 1300
7.32 Rebuilding Primary and Standby Nodes 1304
7.32.1 Overview 1304
7.32.2 Technical Principle 1308
7.32.3 Operation Guide 1319
7.32.4 Typical Issues 1321
7.33 Upgrade 1322
7.33.1 In-Place Upgrade (Non-Multi-Primary) 1322
7.33.1.1 Overview 1322
7.33.1.2 Technical Principle 1327
7.33.1.3 Operation Guide 1327
7.33.1.4 Common Issues 1331
7.33.2 Gray Upgrade (Non-Multi-Primary) 1331
7.33.2.1 Overview 1331
7.33.2.2 Technical Principles 1336
7.33.2.3 Operation Guide 1337
7.33.2.4 FAQ 1342
7.33.3 Rolling Upgrade 1342
7.33.3.1 Overview 1342
7.33.3.2 Technical Principle 1343
7.33.3.3 Operation Guide 1346
7.33.3.4 Common Issues 1351
7.34 Repairing and Replacing Nodes 1351
7.34.1 Overview 1351
7.34.2 Technical Principles 1352
7.34.3 Operation Guide 1353
7.34.4 FAQ 1359
7.35 Plan Cache 1359
7.35.1 Prepared Plan Cache 1360

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.35.1.1 Overview 1360
7.35.1.2 Technical Principle 1361
7.35.1.3 Operation Guide 1362
7.35.1.4 FAQ 1365
7.35.2 Automatic Parameterization and Plan Reuse for Identical Statements 1365
7.35.2.1 Overview 1366
7.35.2.2 Technical Principle 1367
7.35.2.3 Operation Guide 1371
7.35.2.4 FAQ 1373
7.35.3 Global Plan Cache 1374
7.35.3.1 Overview 1374
7.35.3.2 Technical Principles 1375
7.35.3.3 Operation Guide 1379
7.35.3.4 FAQ 1381
7.35.4 Upper Limit of the Plan Cache Memory Control 1382
7.35.4.1 Overview 1382
7.35.4.2 Technical Principles 1383
7.35.4.3 Operation Guide 1386
7.35.4.3.1 Session-Level Plan Cache Control 1386
7.35.4.3.2 GPC Control 1387
7.35.4.4 FAQ 1389
7.36 DATABASE LINK 1389
7.36.1 Overview 1389
7.36.2 Technical Principle 1391
7.36.3 Operation Guide 1394
7.36.4 Typical Issues 1407
7.37 Single-Node Table 1407
7.37.1 Overview 1407
7.37.2 Technical Principles 1407
7.37.3 Operation Guide 1409
7.37.3.1 DDL Operations on Single-Node Tables 1410
7.37.3.2 Online Conversion of Single-Node Tables 1413
7.37.3.3 DML Operations on Single-Node Tables 1415
7.37.4 FAQ 1420
7.38 ROWID Table 1421
7.38.1 Overview 1421
7.38.2 Operation Guide 1421
7.38.2.1 DDL 1422
7.38.2.2 ROWID Type 1423
7.38.2.3 DML 1423
8 Basic Concepts 1463
8.1 Database Objects 1463

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.2 Schema 1468
8.3 Constraints 1470
8.4 Sequences 1472
8.5 System Columns 1474
8.6 Partitioned Tables 1474
8.7 Storage Engine 1477
8.8 Index 1489
8.9 Materialized Views 1492
8.10 Transactions 1494
8.11 Locks 1497
8.12 PL/SQL 1499
8.12.1 Introduction to PL/SQL 1499
8.12.2 Cursors 1500
8.12.3 Stored Procedures 1503
8.12.4 Functions 1505
8.12.5 Trigger 1505
8.12.6 Packages 1507
8.13 Flashback 1509
8.14 VACUUM 1517
8.15 TOAST Table 1526
8.16 HTAP 1528
8.17 Fully-Encrypted Database 1531
8.18 DDL 1535
8.19 Write Ahead Logs 1537
8.20 Ultimate RTO 1542
8.21 DCF 1544
8.22 CM 1548
8.23 Call Stack 1551
8.24 Wait Events 1554
8.25 Backup and Restoration 1557
9 FAQ - Non-Multi-primary 1561
9.1 Database System Overview 1561
9.1.1 Which Hardware Platforms Are Supported by GaussDB? 1561
9.1.2 How Do I Check the Isolation Level of the Current Database? 1561
9.1.3 How Easy-to-Use Is GaussDB? 1561
9.1.4 How Do I Create and Delete a Savepoint in the Current Transaction of GaussDB? 1562
9.1.5 What Are the Functions of a Transaction? 1562
9.1.6 How Do I Start a Transaction? 1563
9.1.7 What Are the Four Features of Correctly Executing Database Transactions? 1563
9.1.8 What is Dirty Read, Phantom Read, and Non-repeatable Read? 1564
9.1.9 What Is MVCC? 1564
9.2 Database Connection Management 1564

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.2.1 How Do I Query the Number of Active Database Connections?....1564
9.2.2 What Are the Causes of the Failure to Connect to the Database Through the Client?....1564
9.2.3 What Client Access Authentication Modes Does GaussDB Support?....1565
9.2.4 How Do I Check the Number of Database Application Connections and the Maximum Number of Connections?....1565
9.2.5 How do I obtain the username for connecting to a database?....1565
9.2.6 How Do I Check the Maximum Number of Session Connections for Specific Users?....1566
9.3 Database Security....1566
9.3.1 How Do I Manually Lock and Unlock a User?....1566
9.3.2 What Security Management Functions Are Supported in GaussDB?....1566
9.3.3 How Do I Generate an OBS Transfer Key Before Creating a Model Service?....1567
9.3.4 How Does a Dynamic Data Masking Policy Take Effect?....1567
9.3.5 How Do I Set the Validity Period of an Account?....1567
9.3.6 How Do I Enable Audit Policies on GaussDB and What Contents Can Be Audited?....1567
9.3.7 How Do I View the Table Permissions of a User?....1568
9.3.8 How Do I Enable Separation of Duties for GaussDB?....1569
9.3.9 How Do I Create a Read-only User on GaussDB?....1569
9.3.10 What Are the Session States of GaussDB? How Do I Query the Session Status of a User?....1570
9.3.11 How Do I Query the Expiration Date of an SSL Certificate?....1570
9.3.12 How Do I Enable Password Complexity Verification?....1571
9.3.13 How Do I Check the Initialized User Role of a Database?....1571
9.3.14 Which System Views Are Related to Audit Information?....1571
9.3.15 How Do I Configure Log File Permissions?....1572
9.3.16 How Do I Set the Maximum Number of Failed Login Attempts for a Database Account?....1572
9.3.17 What Are the Differences Between TDE and Full Encryption?....1572
9.3.18 How Do We Ensure Data Security?....1572
9.3.19 How Do I Create a Weak Password in GaussDB?....1573
9.3.20 What Are the Schemas, Users, and Roles in GaussDB?....1573
9.3.21 Why Can a User Log In to the Database After the CONNECT ON DATABASE Permission Is Granted to the REVOKE User?....1573
9.3.22 How Do I Check the Creation Time of a Database User?....1574
9.4 Database Quick Start....1574
9.4.1 What are the differences between the DROP, TRUNCATE, and DELETE methods in GaussDB?....1574
9.4.2 How many bytes does a Chinese character occupy in GaussDB?....1575
9.4.3 How Do I Create Indexes in Parallel?....1575
9.4.4 How Do I Check the Character Encoding Types of the GaussDB Client and Server?....1575
9.4.5 How Do I View the Current Session ID and Login Time?....1575
9.4.6 How Do I View the Creation Time of a Table?....1576
9.4.7 How Do I View Database Parameter Values?....1576
9.4.8 How Do I View Foreign Table Information?....1576
9.4.9 How Do I View the SELECT Statements That Are Running in the System?....1577
9.4.10 How Do I Query Indexes in a Table?....1577
9.4.11 How Do I Query the Storage Mode of a Table?....1577

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.4.12 How Do I Query the Table and Database Capacity?.....1578
9.4.13 How Do I Query the Number of Sessions?.....1578
9.4.14 How Do I Query the Sizes of Databases, Tablespaces, and Tables?.....1578
9.4.15 How Do I Query the Database Running Time?.....1579
9.4.16 How Do I Query the Last Restart Time of a Database?.....1579
9.4.17 How Do I Create a Column with the Current Time As the Default Value?.....1579
9.4.18 How Do I Create a Read-only View and Specify Whether the View Is Read-only?.....1579
9.4.19 How Do I Disable the System Catalog Structure Modification?.....1579
9.4.20 How Do I Determine Whether a Column is NULL?.....1580
9.4.21 How Do I Delete Duplicate Rows?.....1580
9.4.22 How Do I Use SQL Statements to View the Database Mode?.....1581
9.4.23 How Do I Query the Definitions of Tables, Indexes, and Views Using SQL Statements?.....1581
9.4.24 What Are the Advantages of Data Partitioning?.....1581
9.4.25 What Are the Differences Between UNIQUE Constraints and Unique Indexes?.....1582
9.4.26 Why Cannot the Table Name or Column Name Be Identified During Query?.....1582
9.4.27 What Is the Schema Starting with pg_temp or pg_toast_temp?.....1583
9.4.28 After the String Processing Function Is Used, the Query Does Not Use the Index. What Should I Do?.....1583
9.5 Development Design.....1584
9.5.1 What Is the Maximum Number of Columns in a Single GaussDB Table?.....1584
9.5.2 What Are the Types of SQL Languages?.....1585
9.5.3 What Is the Functionality of a Trigger?.....1585
9.5.4 What Are the Differences Between Functions and Stored Procedures?.....1585
9.5.5 How Do I Query the Triggers in the Current Database and Their Definitions?.....1586
9.5.6 How Do I Query Partition Information and Index Information of a Partitioned Table?.....1587
9.5.7 How Do I Query All Triggers Involved in a Table?.....1587
9.5.8 Is There a Limit on the Length of a Database Object Name?.....1587
9.5.9 Which of the Following Factors Should Be Considered When You Select a Database Character Set?.....1587
9.6 Others.....1587
9.6.1 How Do I Check the Disk Type?.....1588
9.6.2 How Do I Query the Active Tuples, Dirty Tuples, and Dirty Tuple Ratios?.....1588
9.6.3 What Are the Main Differences Between VACUUM and VACUUM FULL?.....1588
9.6.4 What Are the Differences Between pg_terminate_backend and pg_terminate_session in GaussDB?.....1589
9.6.5 How Do I Display Database File Information?.....1589
9.6.6 How Often Are Xlogs in GaussDB Automatically Recycled?.....1590
9.6.7 What Is a Dead Tuple?.....1590
9.6.8 How Do I Query Dead Tuples of a Table?.....1590
9.6.9 What Is OID?.....1590
9.6.10 What Is UDF?.....1591
9.6.11 What Is the Depth of Nested Calling of a Stored Procedure?.....1591
9.6.12 What Is TOAST?.....1591

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.13 What Is the Function of the Sandbox? 1591
9.6.14 What Is a Vector Database? 1591
9.6.15 What Is a GIN Index? 1592
9.6.16 Can a Stored Procedure Call a Trigger? 1592
9.6.17 How Do I Query Several Random Rows in a Table? 1592
9.6.18 Why Is There a Gap in the Sequence Number? 1592
9.6.19 What Do I Do If the Result of the power() Function Is Too Large? 1593
9.6.20 What Is CTID? 1593
9.6.21 How Do I Query the Mth Largest Data in a Column of a Table? 1594
9.6.22 How Do I Query Data Whose Columns Contain Non-digit Characters? 1594
9.6.23 How Do I Change the Column Sequence? 1594
9.6.24 What Is the Difference Between count(*) and count(column)? 1595
9.6.25 What Wildcards Are Supported in GaussDB? How Do I Use Them? 1595
9.6.26 What Are the Differences Between LIMIT 2, LIMIT 2,3, and LIMIT 2 OFFSET 3? 1596
9.6.27 What Is the Relationship Between an Empty String and NULL? 1596
9.6.28 Which Date/Time Types Contain the Time Zone and Which Do Not? 1596
9.6.29 If No Distribution Key Is Specified, How Will Data Be Stored? 1597
9.6.30 What Is Data Skew and How Do I Check Data Skew? 1598
9.6.31 How Do I Ignore Case When Using LIKE for Fuzzy Search? 1598

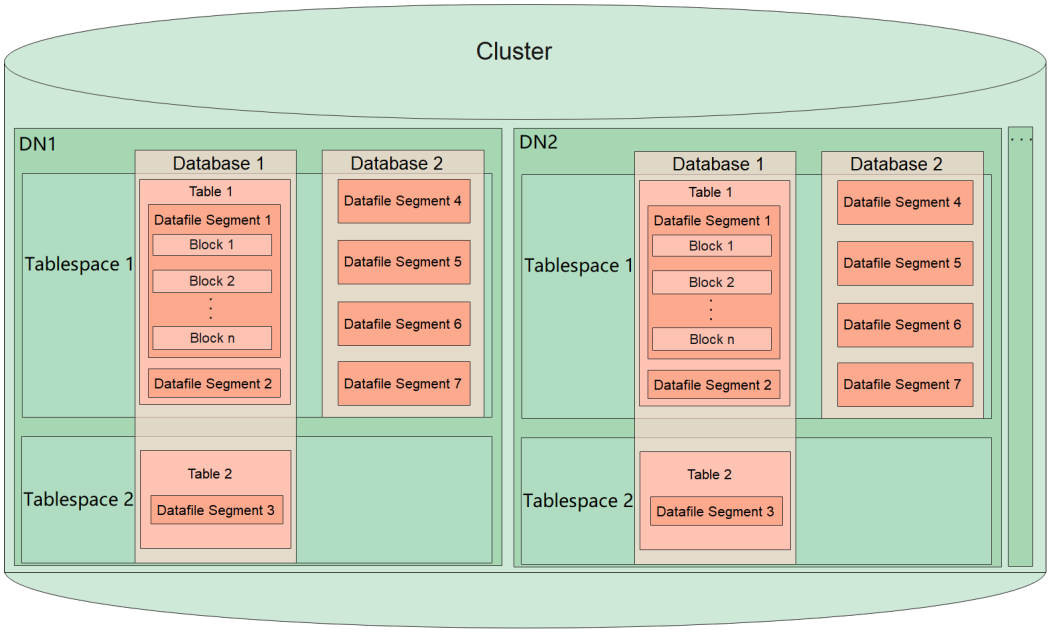
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1 Database System Overview

1.1 Database Logical Architecture

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

Figure 1-1 Database logical architecture



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

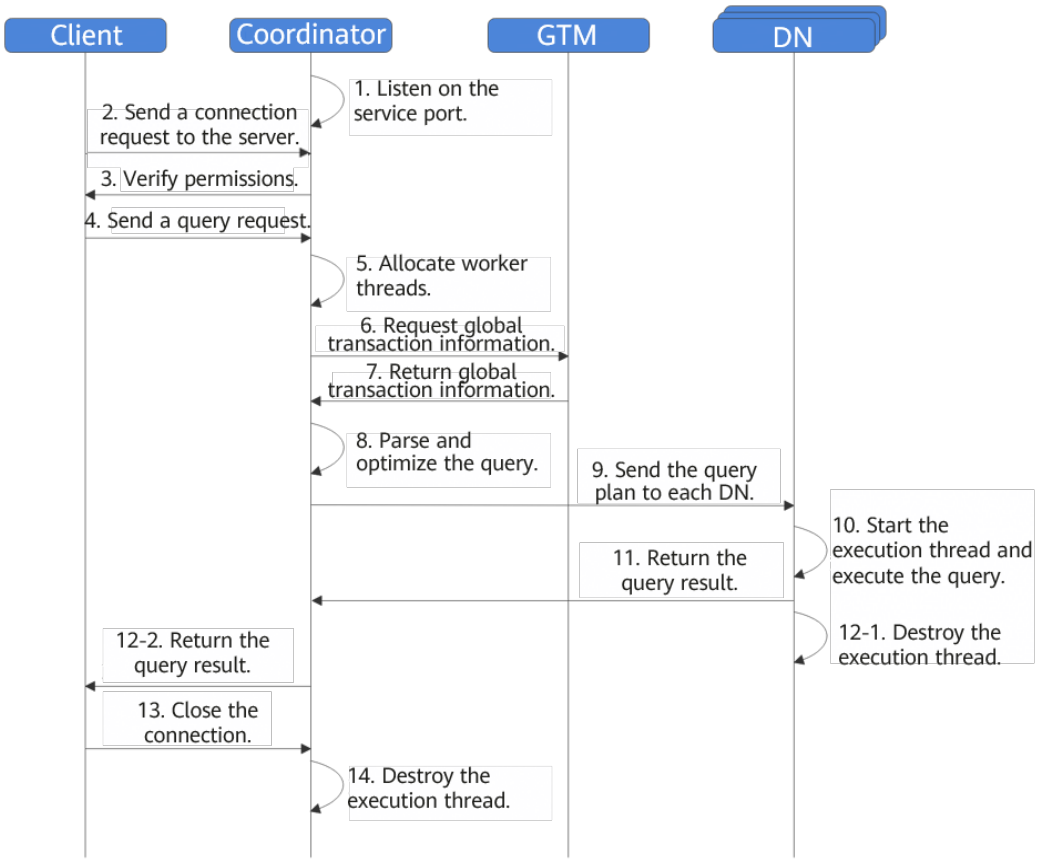
- A tablespace is a directory. A cluster contains one or more tablespaces to store physical files of the database. Each tablespace can contain files belonging to different databases.
- A database manages various data objects and is isolated from each other. Objects managed by a database can be distributed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

1.2 Query Request Handling Process

GaussDB's service response process involves these components: client, coordinator, Global Transaction Manager (GTM), and data node (DN).

Figure 1-2 shows the specific process.

Figure 1-2 GaussDB's service response process



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1.3 Managing Transactions

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

Controlling Transactions

The following describes transaction operations supported by the database:

- Starting transactions

  You can use the START TRANSACTION or BEGIN syntax to start a transaction.
- Setting transactions

  You can use the SET TRANSACTION or SET LOCAL TRANSACTION syntax to set transactions.
- Committing transactions

  You can use the COMMIT or END syntax to commit all operations in a transaction.
- Rolling back transactions

  If faults block an ongoing transaction, the system will undo all operations that the transaction has performed on the database.

In non-multi-primary mode, consult the "START TRANSACTION", "BEGIN", "SET TRANSACTION", "COMMIT | END", and "ROLLBACK" sections under "SQL Reference > SQL Syntax" in *Reference* for details about transaction operations.

Transaction Isolation Levels

The transaction isolation level controls the processing behavior of different isolation level data when multiple transactions access the same data resource.

NOTE

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

- **READ COMMITTED**: At this level, a transaction can access only committed data. This is the default level.

  The SELECT statement accesses the snapshot of the database taken when the query begins. The SELECT statement can also access the data modifications in its transaction, regardless of whether they have been committed. Note that different database snapshots may be available to two consecutive SELECT statements for the same transaction, because data may be committed for other transactions while the first SELECT statement is executed.

  At the **READ COMMITTED** level, the execution of each statement begins with a new snapshot, which contains all the transactions that have been committed by the execution time. Therefore, during a transaction, a statement can access the result of other committed transactions. Check whether a single statement always accesses absolutely consistent data in a database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Transaction isolation at this level meets the requirements of many applications, and is fast and easy to use. However, applications performing complicated queries and updates may require data that is more consistent than this level can provide.

- **REPEATABLE READ**: At this level, a transaction can only read data committed before it starts. Uncommitted data or data committed in other concurrent transactions cannot be read. However, a query can read earlier data modifications in its transaction, regardless of whether they have been committed. **READ COMMITTED** differs from this level in that a transaction reads the snapshot taken at the start of the transaction, not at the beginning of the current query within the transaction. Therefore, the SELECT statement within a transaction always reads the same data, and cannot read data committed by other concurrent transactions after the transaction starts. Applications at this level must be able to retry transactions, because serialization failures may occur.

  REPEATABLE READ uses multi-version snapshots, which can lead to write skew. To prevent this, perform a SELECT FOR UPDATE on the rows involved in the transaction beforehand. The following is an example of write skew:

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

```
gaussdb=# CREATE TABLE a(id int, value int);
CREATE TABLE
gaussdb=# INSERT INTO a VALUES(1,4);
INSERT 0 1
gaussdb=# INSERT INTO a VALUES(2,4);
INSERT 0 1
session1 :
gaussdb=# START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# UPDATE a SET value = 6 WHERE id = 1;
UPDATE 1
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |     6
2 |     4
(2 rows)
session2:
gaussdb=# START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# UPDATE a SET value = 6 WHERE id = 2;
UPDATE 1
gaussdb=# SELECT * FROM a;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
id | value
----+-------
1 |     4
2 |     6
(2 rows)
session1:
gaussdb=# COMMIT;
COMMIT
session2:
gaussdb=# COMMIT;
COMMIT
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |     6
2 |     6
(2 rows)
```

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

```
gaussdb=# CREATE TABLE a(id int PRIMARY KEY, value int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "a_pkey" for table "a"
CREATE TABLE
gaussdb=# INSERT INTO a VALUES(1,10);
INSERT 0 1
session1:
gaussdb=# START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION
gaussdb=# DELETE a WHERE id = 1;
DELETE 1
session2:
gaussdb=# START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |    10
(1 row)
session1:
gaussdb=# COMMIT;
COMMIT
session2:
gaussdb=# INSERT INTO a VALUES(1, 100);
INSERT 0 1
gaussdb=# SELECT * FROM a;
id | value
----+-------
1 |    10
1 |   100
(2 rows)
```

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2 Database Security

2.1 Users and Permissions

2.1.1 Default Permission Mechanism

When a user creates an object, they become the owner of that object. After a cluster is installed, database system administrators have the same permissions as object owners by default. After an object is created, only the object owner and system administrators can query, modify, or destroy it, and grant its permissions to other users using GRANT. For details, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.

To allow other users to use the object, you must grant the necessary permissions to the target users or their roles.

GaussDB supports the following permissions: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, CREATE, CONNECT, EXECUTE, USAGE, ALTER, DROP, COMMENT, INDEX, and VACUUM. Different permissions are associated with different object types. For details about these permissions, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.

To revoke granted permissions, refer to "SQL Reference > SQL Syntax > R > REVOKE" in *Reference*. Object owners implicitly receive permissions such as ALTER, DROP, COMMENT, INDEX, VACUUM, GRANT, and REVOKE on their objects. In other words, the system grants these implicit permissions as soon as a user becomes an object owner. An object's owner can choose to revoke their own ordinary privileges (SELECT, INSERT, UPDATE, and DELETE), for example, to make a table read-only for themselves as well as others (excluding system administrators).

System catalogs and views are either visible only to system administrators or visible to all users. Only system administrators can query the system catalogs and views marked as requiring SYSADMIN permissions. For details, see "System Catalogs and System Views" in *Reference*.

The database provides the object isolation feature. When enabled, it limits users to viewing only the objects (tables, views, columns, and functions) they have

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

permission to access. System administrators are not affected by this feature. For details, see "SQL Reference > SQL Syntax > A > ALTER DATABASE" in *Reference*.

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

2.1.2 Administrators

Initial User

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

NOTE

- The OID of the initial user is 10, which can be queried from the gs_roles view.
- An initial user bypasses all permission checks. It is recommended that this user be used only as a database administrator for database management instead of service applications.

System Administrator

A system administrator is an account with the SYSADMIN attribute. By default, a system administrator has the same permissions as the object owner but does not have the object permissions in the dbe_perf schema.

To create a system administrator, connect to the database as the initial user or a system administrator and use a statement with the **SYSADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

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

Or

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

When ALTER USER is executed, the user must exist.

Security Administrator

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

of duties is enabled, connect to the database as a security administrator and use a statement with the **CREATEROLE** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

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

Or

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

When ALTER USER is executed, the user must exist.

Audit Administrator

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

If you want to create an audit administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database only as the initial user and use a statement with the **AUDITADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

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

Or

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

When ALTER USER is executed, the user must exist.

Monitor Administrator

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

To create a monitor administrator, connect to the database as a system administrator and use a statement with the **MONADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

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

Or

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

When ALTER USER is executed, the user must exist.

O&M Administrator

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

To create an O&M administrator, connect to the database as the initial user and use a statement with the **OPRADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

Or

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

When ALTER USER is executed, the user must exist.

Security Policy Administrator

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

To create a security policy administrator, connect to the database as a system administrator and use a statement with the **POLADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

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

Or

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

When ALTER USER is executed, the user must exist.

2.1.3 Users

You can use CREATE USER and ALTER USER to create and manage database users, respectively. In non-multi-primary mode, a database cluster may contain one or more databases. Users and roles are shared within the entire cluster, but their data is not shared. That is, a user can connect to any database, but once successfully connected, they can access only the database specified in their connection request.

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

Creating, Modifying, and Dropping Users

- Execute the CREATE USER statement to create a user. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" in *Reference*.

  For example, create user **joe** with the CREATEDB attribute.

  ```
  gaussdb=# CREATE USER joe WITH CREATEDB PASSWORD "********";
  CREATE ROLE
  ```

- To create a system administrator, use a statement with the **SYSADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" in *Reference*.

  ```
  gaussdb=# CREATE USER tom WITH SYSADMIN PASSWORD "********";
  CREATE ROLE
  ```

- To drop an existing user, refer to "SQL Reference > SQL Syntax > D > DROP USER" in *Reference*.

  ```
  gaussdb=# DROP USER tom;
  DROP ROLE
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- To modify a user (for example, rename a user or change its password), refer to "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.
  ```
  gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';
  ALTER ROLE
  ```
- Query the user list from the gs_user view. For details, see "System Catalogs and System Views > System Views > User and Permission Management > GS_USER" in *Reference*.
  ```
  gaussdb=# SELECT * FROM gs_user;
  ```
- Query user attributes from the pg_authid system catalog. For details, see "System Catalogs and System Views > System Catalogs > Connection and Authentication > PG_AUTHID" in *Reference*.
  ```
  gaussdb=# SELECT * FROM pg_authid;
  ```

2.1.4 Roles

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

GaussDB provides an implicitly defined group **PUBLIC** that contains all roles. By default, all new users and roles have the permissions of **PUBLIC**. For details about the default permissions granted to **PUBLIC**, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*. 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 query all roles, access the GS_ROLES system view. For details, see "System Catalogs and System Views > System Views > User and Permission Management > GS_ROLES" in *Reference*.

```
SELECT * FROM GS_ROLES;
```

Creating, Modifying, and Dropping Roles

In scenarios without separation of duties, only system administrators and users with the CREATEROLE attribute can create, modify, and drop roles.

- To create a role, refer to "SQL Reference > SQL Syntax > C > CREATE ROLE" in *Reference*.
- To add or drop users in an existing role, refer to "SQL Reference > SQL Syntax > A > ALTER ROLE" in *Reference*.
- To drop a role, refer to "SQL Reference > SQL Syntax > D > DROP ROLE" in *Reference*. DROP ROLE drops only a role, rather than member users in the role.

Built-in Roles

GaussDB provides a group of default roles whose names start with **gs_role_**. These roles are provided to access to specific, typically high-privileged operations. You can grant these roles to other users or roles within the database so that they can use specific functions. Be careful when granting these roles to ensure that they are used where needed. **Table 2-1** describes the permissions of built-in roles.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 2-1 Permissions of built-in roles

Role	Permission
gs_role_signal_backend	This role has permission to call the pg_cancel_backend, pg_terminate_backend, and pg_terminate_session functions to cancel or terminate other sessions. It can also call pg_terminate_active_session_socket to close active sessions and client socket connections. However, it cannot deal with sessions owned by the initial user or the **PERSISTENCE** user.
gs_role_tablespace	This role has permission to create tablespaces.
gs_role_replication	This role has permission to call functions related to logical replication.
gs_role_account_lock	This role has permission to lock or unlock users, excluding the initial user and the **PERSISTENCE** user.
gs_role_pldebugger	This role has permission to debug functions in **dbe_pldebugger**.
gs_role_public_dblink_drop	This role has permission to drop public database links.
gs_role_public_dblink_alter	This role has permission to modify public database links.
gs_role_seclabel	This role has permission to create, delete, and apply security labels.
gs_role_public_synonym_create	This role has permission to create public synonyms.
gs_role_public_synonym_drop	This role has permission to drop public synonyms.

The restrictions on built-in roles are as follows:

- The role names starting with **gs_role_** are reserved for built-in roles in the database. Do not create users, roles, or schemas starting with **gs_role_** or rename existing users, roles, or schemas to names starting with **gs_role_**.
- Do not perform ALTER or DROP operations on built-in roles.
- By default, built-in roles do not have the LOGIN permission and do not have preset passwords.
- The gsql meta-commands **\du** and **\dg** do not display information about built-in roles. However, if *pattern* is set to a specific built-in role, the information is displayed.
- When separation of duties is disabled, the initial user, users with the SYSADMIN permission, and users with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. When separation of duty is enabled, the initial user and users

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. For example:

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

2.1.5 Schemas

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

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

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

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

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

Creating, Modifying, and Deleting Schemas

- To create a schema, refer to "SQL Reference > SQL Syntax > C > CREATE SCHEMA" in *Reference*. By default, the initial user and system administrators can create schemas. Other users can create schemas only in databases where they have been granted the CREATE permission. For details about how to grant the permission, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.
- To modify the name or owner of a schema, refer to "SQL Reference > SQL Syntax > A > ALTER SCHEMA" in *Reference*. A schema can be modified by its owner.
- To drop a schema and its objects, refer to "SQL Reference > SQL Syntax > D > DROP SCHEMA" in *Reference*. A schema can be dropped by its owner.
- 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 within the first schema on the **Search Path**.
- To view the owner of a schema, perform the following join query on the PG_NAMESPACE system catalog and GS_USER system view. For details, see

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

"System Catalogs and System Views > System Catalogs > Other System Catalogs > PG_NAMESPACE" and "System Catalogs and System Views > System Views > User and Permission Management > GS_USER" in *Reference*. Replace *schema_name* in the statement with the name of the schema to be queried.

```
gaussdb=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, gs_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
```

- To list all schemas, use the PG_NAMESPACE system catalog.

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

- To list the tables within a schema, use the PG_TABLES system view. For details, see "System Catalogs and System Views > System Views > Other System Views > PG_TABLES" in *Reference*. For example, the following query will return a table list from the pg_catalog schema:

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

Search Path

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

- To view the current search path, refer to "SQL Reference > SQL Syntax > S > SHOW" in *Reference*.

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

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

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

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

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

2.1.6 User Permissions

- Grant permissions on a specific object to a user. For details, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.

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

  For example, first grant permissions on the tpcds schema to user **joe** and then grant the SELECT permission on the **tpcds.web_returns** table to **joe**.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Grant a role to a user to allow the user to inherit the object permissions of the role.

  a. Create a role.

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

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

  b. Grant object permissions to the role. For details, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.

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

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

  c. Grant the role permissions to a user.

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

     NOTE

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

- Revoke user permissions. For details, see "SQL Reference > SQL Syntax > R > REVOKE" in *Reference*.

2.1.7 Row-Level Security Policy

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

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

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

Example: A table shows data for different users. Each user can only view their own data and cannot access the data of other users.

- In non-multi-primary mode:

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

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

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

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

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

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

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

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- PG_STATISTIC and PG_STATISTIC_EXT store sensitive information about statistical objects, such as high-frequency MCVs. If the permission to query the two system catalogs is granted to a common user after the row-level security policy is created, the common user can still access the two system catalogs to obtain sensitive information in the statistical objects.
- PG_STATS, PG_EXT_STATS, and PG_GTT_STATS are statistics views. They will verify users' SELECT permission on columns. If a common user has the SELECT permission and row-level security is configured for the table being queried, the user may inadvertently access certain column values (not entire rows, as statistics are collected and calculated by column) that are not meant to be accessed by them. This can occur when these values happen to be MCVs in the statistics views.

2.2 Database Audit

Context

Database audit is critical to the security of the database system. Database audit administrators can use the audit logs to reproduce a series of events that cause faults in the database and identify unauthorized users, unauthorized operations, and the time when these operations are performed.

You need to know the following about the audit function:

- The GUC parameter **audit_enabled** is used to enable or disable the audit function. Its value can be dynamically loaded. If you change the configuration when the database is running, the change takes effect immediately and you do not need to restart the database. Its default value is **on**, indicating that the audit function is enabled.
- In addition to the overall audit configuration, each audit item can be independently configured. The function of each audit item takes effect only after the configuration is enabled.
- The configuration of each audit item supports dynamic loading. If you change the audit configuration when the database is running, the change takes effect immediately and you do not need to restart the database.

Table 2-2 describes the audit items supported by GaussDB. If you need to modify a specific audit configuration item, contact an administrator. For details about parameter types and value ranges, see "Database GUC Parameters > GUC Parameters > Auditing" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 2-2 Audit items

Configuration Item	Description
User login and logout audit	Parameter: **audit_login_logout** Its default value is **7**, which indicates that the function of user login and logout audit is enabled. **0** indicates that the function of user login and logout audit is disabled. Other values are not recommended.
Database startup, stop, recovery, and switchover audit	Parameter: **audit_database_process** Its default value is **1**, which enables the audit of database startup, stop, recovery, and switchover. (However, the audit is not supported in multi-primary mode.)
User locking and unlocking audit	Parameter: **audit_user_locked** Its default value is **1**, which indicates that the audit of user locking and unlocking is enabled.
Unauthorized access audit	Parameter: **audit_user_violation** Its default value is **1**, which indicates that the audit of unauthorized user operations is enabled.
Permission granting and revoking audit	Parameter: **audit_grant_revoke** Its default value is **1**, which indicates that the audit of permission granting and revoking is enabled.
Full audit of user operations	Parameter: **full_audit_users** Its default value is an empty string, indicating that the default configuration is used and no full audit of user operations is configured.
Names and IP addresses of clients that do not need to be audited	Parameter: **no_audit_client** Its default value is an empty string, indicating that the default configuration is used and no clients and IP addresses are added to the audit blacklist.
Audit of CREATE, ALTER, and DROP operations on database objects	Parameter: **audit_system_object** In non-multi-primary mode: With the default value of **67121159**, the system audits the CREATE, ALTER, and DROP operations for databases, schemas, users, node groups, and SQL patches.
Audit of INSERT, UPDATE, and DELETE operations on a specific table	Parameter: **audit_dml_state** Its default value is **0**, which indicates that the audit of DML operations (except SELECT) on a specific table is disabled.
SELECT operation audit	Parameter: **audit_dml_state_select** Its default value is **0**, which indicates that the audit of the SELECT operation is disabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Configuration Item	Description
COPY operation audit	Parameter: **audit_copy_exec** Its default value is **1**, which indicates that the audit of COPY operations is enabled.
Audit of execution of stored procedures and user-defined functions	Parameter: **audit_function_exec** Its default value is **0**, which indicates that no execution audit logs of stored procedures and user-defined functions are recorded.
Audit of system functions in the whitelist	Parameter: **audit_system_function_exec** Its default value is **0**, which indicates that audit logs of system function execution are not recorded.
SET operation audit	Parameter: **audit_set_parameter** With the default value of **0**, audit logs are not recorded for the SET operation.
Transaction ID record	Parameter: **audit_xid_info** Its default value is **0**, which indicates that the function of recording transaction IDs in audit logs is disabled.
Audit of internal tool connections and operations, and audit of logins and logouts from CNs on DNs	Parameter: **audit_internal_event** In non-multi-primary mode: The default value is **off**, which means not to audit login/logout events or operations for internal tools (cm_agent, gs_clean, as well as WDRXdb), or CN login/logout events on DNs.

NOTE

If an audit log is recorded when you use EXPLAIN to view an execution plan, the audit type in the log will depend on the specific SQL statement being analyzed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3 Database Quick Start

3.1 Connecting to a Database

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

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

3.1.1 Using gsql to Connect to a Database

gsql is a database connection tool running in the CLI provided by GaussDB. gsql provides basic and advanced functions of databases to facilitate user operations.

Precautions

By default, if a client connects to the database but remains idle for the period specified by the GUC parameter **session_timeout**, the client will be automatically disconnected from the database. To disable the timeout, set **session_timeout** to **0**.

Prerequisites

You have obtained connection information from an administrator.

Remotely Connecting to a Database in Non-Multi-Primary Mode

Step 1 Contact an administrator to configure a remote connection.

Step 2 On the client, upload the client tool package and configure environment variables for gsql.

1. Log in to the client.
2. Create a **tools** folder in the **/tmp** directory. (**/tmp/tools** is only an example—you can use any directory you prefer.)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
mkdir /tmp/tools
```

3. Obtain the tool package **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Gsql.tar.gz** from the software installation package and upload it to **/tmp/tools**.

   NOTE

   - The relative location of the software package is the location specified during installation. Update it to reflect the actual location.
   - The tool package name varies depending on the OS. Select the correct one based on your OS type.

4. Decompress the package.

   ```
   cd /tmp/tools
   tar -zxvf GaussDB-Kernel_Database version number_OS version number_64bit_Gsql.tar.gz
   ```

5. Set environment variables.

   Open the **~/.bashrc** file.

   ```
   vi ~/.bashrc
   ```

   Enter the following content and run **:wq!** to save and exit:

   ```
   export PATH=/tmp/tools/bin:$PATH
   export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH
   ```

6. Make the environment variables take effect.

   ```
   source ~/.bashrc
   ```

Step 3 Connect to a database.

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

- Use the IP address of the server where the CN is located to connect to the database.

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

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

- After a successful connection, the system displays information similar to the following:

  ```
  gsql ((GaussDB Kernel XXX.X.XXX build 9235acb6) compiled at 2025-06-10 07:15:43 last mr 23922 release)
  Non-SSL connection (SSL connection is recommended when requiring high-security)
  Type "help" for help.
  ```

  If **jack** is an administrator, the system displays "DBNAME=#". When a common user logs in and connects to the database, the system displays "DBNAME=>".

  Non-SSL connection indicates that the database is not connected in SSL mode. Use the SSL mode in scenarios demanding high security.

- Connect to the database using an IPv6 address.

  ```
  gsql -d postgres -h 2407:c182:14f0:594:er63:49c5:493q:594c -U jack -p 8000
  ```

- Connect to the database on three CNs.

  In a multi-CN database environment, if you want to skip a faulty CN, you can use gsql to connect to a normal CN.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For example, if there are three CNs in a database instance, and the IP addresses and port numbers of the three CNs are **10.10.0.11:8000**, **10.10.0.12:8002**, and **10.10.0.13:8004**, respectively, you run the following command:

```
gsql -d postgres -h 10.10.0.11,10.10.0.12,10.10.0.13 -U jack -p 8000,8002,8004
```

Gsql connects to the three *IP:port* pairs in sequence. If the currently connected *IP:port* pair is not a normal CN, it automatically moves to the next one until it successfully connects to a normal CN.

You can adjust the value of the environment variable *PGTARGETSESSIONATTRS* to connect to different types of nodes.

NOTE

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

----End

3.1.2 APIs

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

Supported APIs

Each application is an independent GaussDB development project. Applications interact with databases through APIs rather than directly operating on databases. This enhances application portability, scalability, and maintainability. **Table 3-1** lists the APIs supported by GaussDB and the download addresses.

Table 3-1 Database APIs

API	How to Obtain
ODBC	• Linux: Driver: **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Odbc.tar.gz** unixODBC source code package: **https://gitee.com/src-openeuler/unixODBC/blob/openEuler-22.03-LTS-SP1/unixODBC-2.3.7.tar.gz** • Windows: Driver: **GaussDB-Kernel**_*Database version number*_**Windows_Odbc.tar.gz**
JDBC	• Driver: **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Jdbc.tar.gz** • Driver: **com.huawei.gaussdb.jdbc.Driver**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Since JDBC and ODBC APIs connect to databases remotely, GaussDB must be configured to support remote connections. Contact an administrator for related operations.

For details about more APIs, see **Application Development Guide**.

3.2 Operating on a Database

This section describes basic database operations, including creating database users, databases, and tables, inserting data into tables, and querying data in tables. Avoid creating database objects using **rdsAdmin**, as this can lead to service users lacking the necessary permissions to view or operate on these objects. It is advisable to perform such operations with a dedicated service user instead.

3.2.1 Creating a Database User

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

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

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

```
CREATE ROLE
```

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

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

In non-multi-primary mode, run the **GRANT** command to set related permissions. For details, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.

NOTE

For more information about database users, see **Users and Permissions**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.2 Managing Databases in Non-Multi-Primary Mode

Figure 3-1 Creating and managing databases

```
CREATE DATABASE
gaussdb=# \l
                                  List of databases
     Name     | Owner | Encoding  | Collate | Ctype | Access privileges
--------------+-------+-----------+---------+-------+-------------------
 db_test      | omm   | SQL_ASCII | C       | C     |
 db_tpcc      | omm   | SQL_ASCII | C       | C     |
 postgres     | omm   | SQL_ASCII | C       | C     |
 template0    | omm   | SQL_ASCII | C       | C     | =c/omm           +
              |       |           |         |       | omm=CTc/omm
 template1    | omm   | SQL_ASCII | C       | C     | =c/omm           +
              |       |           |         |       | omm=CTc/omm
 template_pdb | omm   | SQL_ASCII | C       | C     |
 templatea    | omm   | SQL_ASCII | C       | C     |
 templatem    | omm   | SQL_ASCII | C       | C     | =c/omm           +
              |       |           |         |       | omm=CTc/omm
 testdb2      | jim   | SQL_ASCII | C       | C     |
(9 rows)

gaussdb=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
ALTER DATABASE
gaussdb=# DROP DATABASE human_tpcds;
DROP DATABASE
gaussdb=#
```

Prerequisites

You must have the permission to create a database or the SYSADMIN permission. For details about how to grant permissions, see **Users and Permissions**.

Context

- GaussDB has four default template databases **template0**, **template1**, **templatea**, and **templatem**, and a default user database **postgres**.
- CREATE DATABASE creates a database by copying a template database By default, **template0** is copied. Do not use clients or other methods to connect to or operate on the two template databases.

  NOTE

  - The template database does not contain any user table. You can view the attributes of the template database in the PG_DATABASE system catalog.
  - The **template0** and **templatea** templates do not allow user connections. Only the initial user of the database and system administrators can connect to **template1** and **templatem**.
  - **templatem** is a database name reserved by the system and cannot be a name of database created by users.
  - The name of **templatea** or **templatem** cannot be changed.
- A database system consists of multiple databases. A client can connect to only one database at a time. Currently, cross-database query or cross-database transaction is not supported.
- You can use the **-d** parameter of the gsql client tool to specify the target database for login. Alternatively, you can run the **\c** command to switch the database after logging in to the database through the client.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

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

To resolve this problem, you need to:

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

Procedure

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

```
gaussdb=# CREATE DATABASE db_tpcds;
CREATE DATABASE
gaussdb=# CREATE DATABASE db_tpcc1 TEMPLATE templatea;
CREATE DATABASE
```

NOTE

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

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

  hr_local indicates the tablespace name. For details about how to create a tablespace, see **Managing Tablespaces**.
- After creating the **db_tpcds** database, you can perform other operations in the default **postgres** database. Alternatively, you can perform the following operations to exit the **postgres** database, connect to the **db_tpcds** database as a new user, and perform operations such as creating tables:

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 2 View databases.

- Run the **\l** meta-command to view the database list of the database system.
  ```
  gaussdb=# \l
  ```
- Query the database list in the pg_database system catalog.
  ```
  gaussdb=# SELECT datname FROM pg_database;
  ```

Step 3 Modify the database.

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

- Rename the database.
  ```
  gaussdb=# ALTER DATABASE db_tpcds RENAME TO human_tpcds;
  ALTER DATABASE
  ```

  NOTE

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

Step 4 Drop the database.

You can run the **DROP DATABASE** command to drop a database. This command removes the database system catalog and permanently deletes the data files on the disk. Only the database owner or users with the SYSADMIN permission can perform the drop operation. If there are active database connections, the drop operation will fail. Ensure that you have connected to other databases before performing the drop operation.

Drop the database.

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

----End

3.2.3 Managing Tablespaces

Figure 3-2 Creating and managing tablespaces

```
gaussdb=# CREATE TABLESPACE tbs_t1 RELATIVE LOCATION 'tablespace/tbs_t1';
CREATE TABLESPACE
gaussdb=# CREATE TABLE foo(i int) TABLESPACE tbs_t1;
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# \db
         List of tablespaces
    Name    | Owner |     Location
------------+-------+------------------
 pg_default | omm1  |
 pg_global  | omm1  |
 tbs_t1     | omm1  | tablespace/tbs_t1
 ts_t1      | omm1  | tablespace/ts_t1
(4 rows)

gaussdb=# ALTER TABLESPACE tbs_t1 RENAME TO fspace;
ALTER TABLESPACE
gaussdb=# DROP TABLE foo;
DROP TABLE
gaussdb=# DROP TABLESPACE fspace;
DROP TABLESPACE
gaussdb=#
```

Context

In non-multi-primary mode:

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

- If the disk partition or tablespace initially allocated to the database is full and the space cannot be logically extended, you can create and use tablespaces in other partitions until the system space is reconfigured.
- Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance.
  - A frequently used index can be stored in a disk having stable performance and high computing speed, such as a solid-state device.
  - A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.
- You can use tablespaces to limit the disk space of databases. If a tablespace shares a partition with other tablespaces, the tablespace will never occupy the space allocated to other tablespaces.
- You can use tablespaces to manage the disk space used to store database data. The database will switch to the read-only mode when the disk usage of the tablespace reaches the value of the CM server parameter

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

datastorage_threshold_value_check. Once the disk usage drops below the value of **datastorage_threshold_value_check - datastorage_threshold_recover**, the database will be restored to the read/write mode. The CM disk automatic check feature is enabled by default. If disabled, contact an administrator to enable it.

- Each tablespace corresponds to a file system directory. If you have read and write permissions on an empty directory, such as *data directory*/**pg_location/mount1/path1**, you can create an absolute path tablespace in this directory.

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

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

Precautions

In non-multi-primary mode:

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

  This is because user-defined tablespaces are typically intended for use with storage media other than the main storage (device where the default tablespace is situated, like a disk) to isolate I/O resources for different services. In the Huawei Cloud scenario, however, storage devices follow standard configurations and lack additional storage media options. Improper use of user-defined tablespaces can lead to system instability over time and impact overall performance. Therefore, it is advisable to use the default tablespace.

- In the primary/standby environment, do not specify a file system directory with insufficient permissions on the standby node when creating a tablespace.

Procedure

Step 1 Creates a tablespace.

1. Create user **jack** and set the password to *********.

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

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

   ```
   CREATE ROLE
   ```

2. Create a tablespace.

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

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

   ```
   CREATE TABLESPACE
   ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

fastspace is the new tablespace, and *CN/DN data directory*/**pg_location/my_tablespace/tablespace1** is an empty directory on which the user has read and write permissions.

3. Grant the permission of accessing the **fastspace** tablespace to user **jack** as a database system administrator.
   ```
   gaussdb=# GRANT CREATE ON TABLESPACE fastspace TO jack;
   ```
   If the following information is displayed, the GRANT operation is successful:
   ```
   GRANT
   ```

Step 2 Create an object in a tablespace.

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

Objects can be created in a tablespace in either of the following methods (take creating a table as an example):

- Create a table in a specified tablespace.
  ```
  gaussdb=# CREATE TABLE foo(i int) TABLESPACE fastspace;
  ```
  If the following information is displayed, the CREATE operation is successful:
  ```
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  ```
- Run **set default_tablespace** to set the default tablespace and then create a table.
  ```
  --- In this example, fastspace is the default tablespace, and foo2 is the created table.
  gaussdb=# SET default_tablespace = 'fastspace';
  SET
  gaussdb=# CREATE TABLE foo2(i int);
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  ```

Step 3 Query a tablespace.

You can query a tablespace in either of the following methods:

- Check the pg_tablespace system catalog. View all the tablespaces defined by the system and users.
  ```
  gaussdb=# SELECT spcname FROM pg_tablespace;
  ```
- Run the meta-command provided by gsql to query the tablespace.
  ```
  gaussdb=# \db
  ```

Step 4 Query the tablespace usage.

1. Query the current usage of the tablespace.
   ```
   gaussdb=# SELECT pg_tablespace_size('fastspace');
   ```
   Information similar to the following is displayed:
   ```
    pg_tablespace_size
   --------------------
              2146304
   (1 row)
   ```
   2146304 is the size of the tablespace, and its unit is byte.
2. In non-multi-primary mode, calculate tablespace usage using the following formula:

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 5 Modify a tablespace.

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

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

NOTE

Only the tablespace owner and system administrators can modify a tablespace.

Step 6 Drop a tablespace.

1. Drop user **jack**.

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

2. Drop tables **foo** and **foo2**.

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

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

   ```
   DROP TABLE
   ```

3. Drop tablespace **fspace**.

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

   NOTE

   Only the tablespace owner and system administrators can drop a tablespace.

----End

3.2.4 Managing Tables

3.2.4.1 Creating a Table

Context

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

Creating a Table

Create a table.

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

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

```
CREATE TABLE
```

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- By default, new database objects are created in the $user schema. For more details about schemas, see **Managing Schemas**.
- In addition to the created tables, a database contains many system catalogs. These system catalogs contain cluster installation information and information about various queries and processes in GaussDB. You can collect information about the database by querying system catalogs. For details, see **Querying System Catalogs**.
- In non-multi-primary mode, consult "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Reference* for instructions on creating a table.

3.2.4.2 Inserting Data into Tables

A newly created table contains no data. You need to insert data before using the table. This section describes how to use the **INSERT** command to insert one or more rows of data and to insert data from a specified table. Contact an administrator if a large amount of data needs to be imported into tables in batches.

Context

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

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

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

The preceding string function operations and the automatic truncation mechanism may affect each other. For example, if the character sets of the client and server

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

are inconsistent, and both the string function operations and automatic truncation mechanism exist, the system continues to perform secondary truncation after the string processing is complete. As a result, unexpected results occur. For details, see **Table 3-3**.

NOTE

Ultra long strings are truncated only when the database's **DBCOMPATIBILITY** is set to **TD** (compatible with Teradata) and the GUC parameter **td_compatible_truncation** is set to **on**.

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

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

Table 3-3 Examples

No.	1	2	3	4
Character Set on the Server	SQL_ASCII	SQL_ASCII	SQL_ASCII	SQL_ASCII
Character Set on the Client	UTF8	UTF-8	UTF-8	UTF8
Automatic Truncation Enabled	Yes	Yes	Yes	No
Example	gaussdb=# INSERT INTO *table1* VALUES(*1,reverse('123AAA78'),reverse('123AA78'),reverse('123AA78')*);	gaussdb=# INSERT INTO *table1* VALUES(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	gaussdb=# INSERT INTO *table1* VALUES(*3,'87A123','87A123','87A123'*);	gaussdb=# INSERT INTO *table2* VALUES(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*); gaussdb=# INSERT INTO *table2* VALUES(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);
Result	id |a|b|c ----+------+------+------ 1 | 87| 87| 87	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	id |a|b|c ----+--------------------+--------+-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description	A string is reversed on the server and then truncated. Because character sets used by the server and client are different, character A is displayed in multiple bytes on the client and the result is unexpected.	A string is reversed and then automatically truncated. Therefore, the result is unexpected.	The column length in the string type is an integer multiple of the length in client character encoding. Therefore, the result is as expected after truncation.	Similar to the first example, multi-byte characters are different from the original characters after being reversed.

Procedure

Create a table before inserting data into it. For details about how to create a table, see **Managing Tables**.

- Insert a row into the **customer_t1** table.

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

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

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

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

  If you do not know some of the column values, you can omit them. If the INSERT statement does not explicitly specify the column names of the target table, the multiple values to be inserted in the VALUES clause will correspond one-to-one with the columns of the target table based on their order. This means the first value in the VALUES clause will match the first column of the target table, the second value will match the second column, and so forth. Any columns in the target table that do not have corresponding values in the VALUES clause will be automatically filled with default values or NULL. If no value is specified for a column, the column is set to the default value. For example:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

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

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

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

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

- Insert multiple rows into a table.

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

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

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

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

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

NOTE

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

- Use a table alias without AS to insert data into the table.

```
gaussdb=# CREATE TABLE table_name(a int,b text,c timestamp,d float);
gaussdb=# INSERT INTO table_name alias_name VALUES(1,'ha','1997-01-01 10:00:00'::timestamp,1.234);
gaussdb=# INSERT INTO table_name alias_name(b,c,d) VALUES('ha','1997-01-01 10:00:00'::timestamp,1.234);
```

NOTE

- When the INSERT statement is used without AS, the table alias cannot be a keyword (such as SELECT or VALUE) or an expression. The alias must comply with the identifier naming rule.
- When the INSERT statement is used without AS, the table alias does not support this format: INSERT INTO table_name alias_name(alias_name.col1,...,alias_name.coln) VALUES(xxx).
- When the INSERT statement is used without AS, data cannot be inserted to the specified partition.

- Drop the backup table.

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

NOTE

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.4.3 Updating Data in a Table

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

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

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

SQL does not provide a unique identifier for data rows. Therefore, the WHERE clause is used to locate the target row. If a primary key is defined in a table, the primary key value can be used to accurately specify the row to be updated. Otherwise, the update must depend on the combination of service logic conditions.

For details about how to create a table and insert data to it, see **Creating a Table** and **Inserting Data into Tables**.

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

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

The table name can also be qualified with a schema name. Otherwise, the table is queried from the default schema of the database. SET is followed by the column and the new column value. The new column value can be a constant or an expression.

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

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

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

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

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

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Updating Data in a Table in Batches

When you update a table that contains millions of records in batches, the table will be locked if you directly perform the update operation, which significantly affects the performance of other operations. To avoid this, you can use the following methods to optimize batch update operations:

- Updating data in batches

  You can split a large-scale update into multiple small batches. Only some data is updated each time to shorten the time for locking a table at a time.

  For example, the original batch update statement is UPDATE products SET price = price * 1.1 WHERE category = 'Electronics';

  Optimization solution: Split the batch update statement into multiple UPDATE statements by ID and update some data each time.

  ```
  gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id >= 1 and id <= 10000;
  gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id > 10000 and id <= 20000;
  gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id > 20000 and id <= 30000;
  ...
  ```

- Using a temporary table

  If a transaction contains complex queries, you can insert the records to be updated into a temporary table and then update the main table from the temporary table to prevent the data rows from being locked for a long time.

  For example, the original batch update statement is UPDATE products SET price = price * 1.1 WHERE (size = 101 OR size = 102) AND category = 'Electronics' AND amount > 150;

  Optimization solution: Create a temporary table to store the target records that meet the conditions, update the records, and then update the main table from the temporary table.

  ```
  gaussdb=# CREATE TEMPORARY TABLE temp_products AS SELECT id, price * 1.1 as new_price from products WHERE (size = 101 OR size = 102) AND category = 'Electronics' AND amount > 150;
  gaussdb=# UPDATE products p SET price = t.new_price FROM temp_products t where p.id = t.id;
  gaussdb=# DROP TABLE temp_products;
  ```

- Using parallel updates

  In a multi-thread or multi-process environment, you can update records in parallel, that is, multiple threads update different records at the same time. This requires that the records processed by each thread do not overlap.

  For example, the original batch update statement is UPDATE products SET price = price * 1.1 WHERE category = 'Electronics';

  Optimization solution: Split the data into multiple small batches by ID, start multiple sessions to execute the UPDATE operation at the same time, and reduce the total time required for batch operations through parallel updates. Ensure that the tuples updated by each session do not overlap.

  session1:

  ```
  gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id >= 1 and id <= 10000;
  ```

  session2:

  ```
  gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id > 10000 and id <= 30000;
  ```

  session3:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# UPDATE products SET price = price * 1.1 WHERE category = 'Electronics' AND id > 30000;
```

3.2.4.4 Viewing Data

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

3.2.4.5 Deleting Data from a Table

Expired data may need to be deleted from a table being used. In this case, you must delete the entire row from the table.

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

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

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

Run the following command to delete all rows from the table:

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

Or

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

NOTE

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

Drop the created table.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.5 Querying System Catalogs

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

The descriptions of each table in system catalogs and system views—see "System Catalogs and System Views" in *Reference*—specify whether the table is visible to all users or only to the initial user. To query tables that are visible only to the initial user, log in as the initial user.

GaussDB provides the following types of system catalogs and views:

- PG-compatible system catalogs and views, prefixed with PG.
- New system catalogs and views in GaussDB, prefixed with GS.
- ORA-compatible system catalogs and views, prefixed with ALL, DBA, USER, or PV.

Querying Database Tables

Create the following tables in the public schema:

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

Query the PG_TABLES system catalog to locate tables prefixed with **search_table** in the public schema.

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

The result is as follows:

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

Viewing Database Users

Query PG_USER to retrieve a list of all database users or to check the IDs (**USESYSID**) and permissions of specific users.

```
gaussdb=# SELECT * FROM gs_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillsp
acelimit | usemonitoradmin | useoperatoradmin | usepolicyadmin
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+-----------+-----------+----------------+--------
---------+-----------------+------------------+----------------
 omm     |       10 | t           | t        | t         | t       | ******** |          |          | default_pool |      0 |
```

```
|         |         |             |
       | t            | t             | t
(1 row)
```

Viewing and Stopping the Running Query Statements

You can consult the PG_STAT_ACTIVITY view to identify active query statements, as follows:

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

```
gaussdb=# SET track_activities = on;
```

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

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

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

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

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

```
gaussdb=# SELECT datname, usename, state, pid FROM pg_stat_activity WHERE state != 'idle';
```

Step 3 To cancel long-running queries, call the pg_terminate_backend(pid int) function to terminate the session by thread ID (**pid** queried in **Step 2**).

```
gaussdb=# SELECT pg_terminate_backend(140298793514752);
```

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

```
pg_terminate_backend
----------------------
 t
(1 row)
```

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- When the gsql client calls the pg_terminate_backend(pid int) function to terminate the background thread of the current session, the connection behavior varies depending on the user's identity.
  - Initial user: The client remains connected and is automatically reset. ("The connection to the server was lost. Attempting reset: Succeeded." is displayed.)
  - Non-initial user: The client connection is interrupted and fails to be reset. ("The connection to the server was lost. Attempting reset: Failed." is displayed.)

  Reason: The initial user has the password-free login permission, but common users do not have this permission.
- When the thread pool feature is enabled, calling the pg_terminate_backend(pid int) function to terminate inactive background threads cannot terminate idle sessions because they lack thread IDs. In non-thread pool mode, terminated sessions are not automatically reconnected.

----End

3.2.6 Other Operations

3.2.6.1 Managing Schemas

Figure 3-3 Creating and managing schemas

```
commit 8868 last mr 17511 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

gaussdb=# CREATE SCHEMA myschema;
CREATE SCHEMA
gaussdb=# CREATE TABLE myschema.mytable(id int, name varchar(20));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribu
tion column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution c
olumn.
CREATE TABLE
gaussdb=# SET SEARCH_PATH TO myschema, public;
SET
gaussdb=# SHOW SEARCH_PATH;
   search_path
-----------------
 myschema, public
(1 row)

gaussdb=# DROP SCHEMA myschema CASCADE;
NOTICE:  drop cascades to table mytable
DROP SCHEMA
gaussdb=#
```

Context

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- A database cluster can have one or more databases. Users and user groups are shared within the entire cluster, but their data is not shared. Any user who has connected to a server can access only the database specified in the connection request.
- A database can have one or more schemas, and a schema can contain tables and other data objects, such as data types, functions, and operators. One object name can be used in different schemas. For example, both schema1 and schema2 can have a table named **mytable**.
- Different from databases, schemas are not isolated. You can access the objects in a schema of the connected database if you have schema permissions. To manage schema permissions, you need to have knowledge about database permissions.
- You cannot create a schema prefixed with PG_. Such schemas are reserved for the database system.
- Each time a new user is created, the system creates a schema with the same name for the new user in the current database. In other databases, such a schema needs to be manually created.
- When you reference a table using an unqualified name (consisting of only the table name without a schema), the system determines which table is meant by following **search_path**. It always searches the pg_temp and pg_catalog schemas first, regardless of whether they appear in or where they are placed in **search_path**. **search_path** is a list of schema names to look in. The first matching table in it is considered the target. If no match is found, an error is reported. (Even if a table exists, the search will still fail if its schema is not included in **search_path**.) The first schema in **search_path** is referred to as the *current schema*. Aside from being the first schema searched, it is also the default schema for new database objects if the objects are created without declaring a schema name.
- Each database has a pg_catalog schema, which contains system catalogs and all built-in data types, functions, and operators. pg_catalog is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search_path**. This search order ensures that database built-in objects can be found. If you need to create a user-defined object with the same name as a built-in object, you can perform operations in the schema to which the object belongs to avoid name conflicts.

Creating a Schema

For details, see "SQL Reference > SQL Syntax > C > CREATE SCHEMA" in *Reference*.

- Create a schema.

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

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

  ```
  CREATE SCHEMA
  ```

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Specify the owner when creating a schema.
  ```
  gaussdb=# CREATE SCHEMA myschema AUTHORIZATION omm;
  ```
  If the following information is displayed, the **myschema** schema that belongs to the **omm** user is created successfully:
  ```
  CREATE SCHEMA
  ```

Using a Schema

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

- Create the **mytable** table in myschema.
  ```
  gaussdb=# CREATE TABLE myschema.mytable(id int, name varchar(20));
  CREATE TABLE
  ```
- Query all data of the **mytable** table in myschema.
  ```
  gaussdb=# SELECT * FROM myschema.mytable;
  id | name
  ----+------
  (0 rows)
  ```

Schema Search Path

You can set **search_path** to define the schema search order of objects in a database. The first schema in the search path is called the current default schema. If no schema is explicitly specified when an object is created, the object is automatically created in the default schema.

- View the search path.
  ```
  gaussdb=# SHOW search_path;
   search_path
  ----------------
   "$user",public
  (1 row)
  ```
- Set the search path to **myschema, public** to configure the schema search order in the database. The system searches for objects in the myschema schema first. If no object is found, the system continues to search for objects in the public schema.
  ```
  gaussdb=# SET search_path TO myschema, public;
  SET
  ```

Schema Permission Control

By default, a user can only access database objects in their own schema. To access objects in other schemas, the target schema owner must grant the corresponding USAGE permission.

By granting the CREATE permission on a schema to a user, the user can create objects in this schema. By default, all roles have the USAGE permission in the public schema, but common users do not have the CREATE permission in the public schema. If a common user has the permission to connect to a specified database and create objects in the public schema, security risks may occur. If the common user has the CREATE permission on the public schema, you are advised to run the following statement to revoke the permission:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Revoke **PUBLIC**'s permission to create objects in the public schema. In the following statement, the first **public** indicates the schema and the second **public** indicates all predefined user roles:
```
gaussdb=# REVOKE CREATE ON SCHEMA public FROM public;
REVOKE
```
- Run the following command to view the current schema:
```
gaussdb=# SELECT current_schema();
 current_schema
----------------
 myschema
(1 row)
```
- Create user **jack** and grant the usage permission on **myschema** to the user.
```
gaussdb=# CREATE USER jack IDENTIFIED BY '********';
CREATE ROLE
gaussdb=# GRANT USAGE ON SCHEMA myschema TO jack;
GRANT
```
- Revoke the usage permission on **myschema** from **jack**.
```
gaussdb=# REVOKE USAGE ON SCHEMA myschema FROM jack;
REVOKE
```

Dropping a Schema

For details, see "SQL Reference > SQL Syntax > D > DROP SCHEMA" in *Reference*.

- If a schema is empty, that is, it contains no database objects, you can run the **DROP SCHEMA** command. For example, drop a null schema named **nullschema**.
```
gaussdb=# DROP SCHEMA IF EXISTS nullschema;
DROP SCHEMA
```
- To drop a schema that is not null, use the keyword CASCADE to drop it and all its objects. For example, drop **myschema** and all its objects.
```
gaussdb=# DROP SCHEMA myschema CASCADE;
DROP SCHEMA
```

After the schema is dropped, run the following command to drop user **jack**:
```
gaussdb=# DROP USER jack;
DROP ROLE
```

3.2.6.2 Managing Partitioned Tables

Context

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

- High query performance: You can search for only the partition data that matches the search criteria, improving the search efficiency.
- High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
- Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.
- Balanced I/O: In non-multi-primary mode, partitions can be mapped to different disks to balance I/O, thereby improving the overall system performance.

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

Procedure

This section uses range partitioning as an example to describe related operations. For other partitioning examples, refer to "SQL Reference > SQL Syntax > C > CREATE TABLE PARTITION" in *Reference*. Before performing the operations in this section, ensure that you have created the **tpcds** schema.

Example 1: Use the default tablespace.

- Create a partitioned table.

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

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

```
CREATE TABLE
```

- Insert data.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Create the backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table in the database and insert the data in the **tpcds.customer_address** table to the **tpcds.web_returns_p2** table.

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

- Modify the row movement attributes of the partitioned table.

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

- Drop a partition.

  Drop partition **P8**.

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

- Add a partition.

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

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

- Rename a partition.

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

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

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

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

- Query a partition.

  Query partition **P6**.

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

- Drop partitioned tables and tablespaces.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example 2: Use a user-defined tablespace.

Perform the following operations on a range partitioned table:

- Create tablespaces.

```
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'example1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'example2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'example3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'example4';
```

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

```
CREATE TABLESPACE
```

- Create a partitioned table.

```
gaussdb=# CREATE TABLE tpcds.customer_address
(
    ca_address_sk       integer               NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                     ,
    ca_street_name      character varying(60)               ,
    ca_street_type      character(15)                    ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)             ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                     ,
    ca_zip              character(10)                    ,
    ca_country          character varying(20)               ,
    ca_gmt_offset       numeric(5,2)                      ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

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

```
CREATE TABLE
```

- Insert data.

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

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    ca_gmt_offset       numeric(5,2)                ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- Modify the row movement attributes of the partitioned table.

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

- Drop a partition.

  Drop partition **P8**.

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

- Add a partition.

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

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

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

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

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

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

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

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

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

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

- Query a partition.

  Query partition **P6**.

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

- Drop partitioned tables and tablespaces.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.6.3 Managing Indexes

Context

Indexes accelerate data access but increase the processing time of insertion, update, and deletion operations. Therefore, you need to evaluate the necessity of creating an index and decide on the selection of the index column. You can determine whether to create an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be collated.

Indexes are created based on columns in database tables. Therefore, when creating an index, you need to evaluate the selection criteria of the target column. You can evaluate the necessity of creating an index based on the following conditions and select the column for which the index is to be created:

- Column that is often searched and queried. This speeds up searches.
- Column that is used as the primary key. The primary key must be unique and sorted in order.
- Column that is often joined. This increases the join efficiency.
- Column that is often searched by range. The index helps collate data, and therefore the specified ranges are contiguous.
- Column that often needs to be collated. The index helps collate data, reducing the time for a collation query.
- Column where the WHERE clause is executed frequently. This speeds up condition judgment.
- Column that often appears after the keywords ORDER BY, GROUP BY, and DISTINCT.

  NOTE

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

Procedure

Create a partitioned table before performing operations. For details, see **Managing Partitioned Tables**.

- Creating an index

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Create the partitioned table index **tpcds_web_returns_p2_index1** without specifying the partition name.
    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2 (ca_address_id) LOCAL;
    ```
    If the following information is displayed, the CREATE operation is successful:
    ```
    CREATE INDEX
    ```
  - Create the partitioned table index **tpcds_web_returns_p2_index2** with the partition name specified.
    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2 (ca_address_sk) LOCAL
    (
        PARTITION web_returns_p2_P1_index,
        PARTITION web_returns_p2_P2_index TABLESPACE example3,
        PARTITION web_returns_p2_P3_index TABLESPACE example4,
        PARTITION web_returns_p2_P4_index,
        PARTITION web_returns_p2_P5_index,
        PARTITION web_returns_p2_P6_index,
        PARTITION web_returns_p2_P7_index,
        PARTITION web_returns_p2_P8_index
    ) TABLESPACE example2;
    ```
    If the following information is displayed, the CREATE operation is successful:
    ```
    CREATE INDEX
    ```

- Change the tablespace of an index partition.
  - Change the tablespace of index partition **web_returns_p2_P2_index** to **example1**.
    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P2_index TABLESPACE example1;
    ```
    If the following information is displayed, the ALTER operation is successful:
    ```
    ALTER INDEX
    ```
  - Change the tablespace of index partition **web_returns_p2_P3_index** to **example2**.
    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P3_index TABLESPACE example2;
    ```
    If the following information is displayed, the ALTER operation is successful:
    ```
    ALTER INDEX
    ```

- Rename an index partition.

  Rename the index partition **web_returns_p2_P8_index** to **web_returns_p2_P8_index_new**.
  ```
  gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
  ```
  If the following information is displayed, the RENAME operation is successful:
  ```
  ALTER INDEX
  ```

- Query indexes.
  - Query all indexes defined by the system and users.
    ```
    gaussdb=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
    ```
  - Query information about a specified index.
    ```
    gaussdb=# \di+ tpcds.tpcds_web_returns_p2_index2
    ```

- Drop indexes.
  ```
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

```
DROP INDEX
```

Introduction to Indexes

GaussDB supports four methods for creating indexes. For details, see **Table 3-4**.

Table 3-4 Indexing methods

Indexing Method	Description
Unique index	An index that requires the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, GaussDB automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, unique indexes can be created only for the B-tree and UB-tree in GaussDB.
Composite index	An index that can be defined for multiple attributes of a table. In GaussDB, B-tree supports composite indexes.
Partial index	An index built over a subset of a table. The index contains only tuples that satisfy the relevant conditional expression.
Expression index	An index that is built on a function or expression calculated based on one or more attributes of a table. An expression index works only when the expression used during query is the same as that used during creation.

- Create an ordinary table.

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

- Create an ordinary index.

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

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

  When the database system queries the **tpcds.customer_address_bak** table, if the WHERE condition is matched row by row in full table scan mode, the operation will cause a significant performance bottleneck in the scenario where the table contains a large amount of data but only a few rows (even zero) are matched. If the database system maintains an index on the **ca_address_sk** attribute to quickly locate matched tuples, the database system only needs to traverse a small number of index nodes to obtain matched tuples, greatly improving the query efficiency. Furthermore, indexes can improve the update and deletion operation performance in the database.

  Create an index.

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

- Create a unique index.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

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

- Create a composite index.

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

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

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

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

- Create a partial index.

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

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

- Create an expression index.

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

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

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

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

- Drop the **tpcds.customer_address_bak** table.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2.6.4 Managing Views

Figure 3-4 Creating and managing views

```
gaussdb=# CREATE TABLE mytable (c1 int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE OR REPLACE VIEW myview AS SELECT * FROM mytable WHERE trunc(c1) > 1000;
CREATE VIEW
gaussdb=# SELECT * FROM myview;
 c1
----
(0 rows)

gaussdb=# \d+ myview
                View "public.myview"
 Column |  Type   | Modifiers | Storage | Description
--------+---------+-----------+---------+-------------
 c1     | integer |           | plain   |
View definition:
 SELECT  *
   FROM mytable
  WHERE trunc(mytable.c1::double precision) > 1000::double precision;

gaussdb=# DROP VIEW myview;
DROP VIEW
gaussdb=# DROP TABLE mytable
```

Context

If you need to frequently query a combination of specific columns in multiple tables, you can define a view to avoid repeated query statements. A view abstracts the underlying table structure through a virtual table to simplify common query operations.

A view is a logical table. Different from a physical base table, a view does not actually reside on disks. The database stores only the query definition of the view. The actual data is always stored in the joined base table. When data in the base table changes, the view query result dynamically synchronizes the changes. The view is like a real-time data window, allowing users to access the mirrored data they are concerned about. Each time a view is accessed, the system re-executes its definition query to obtain the latest data.

NOTE

For details about the advantages and disadvantages of views, see **Views**.

Managing Views

Create a table before proceeding. For details, see **Creating a Table**.

- Create a view.

  Create a view named **myview**. The command uses a created user table named **mytable**, which includes the integer column **c1**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE OR REPLACE VIEW myview AS SELECT * FROM mytable WHERE trunc(c1) > 1000;
CREATE VIEW
```

> **NOTE**
>
> The OR REPLACE clause in CREATE VIEW is optional. If this clause is used, the database system automatically replaces the original view definition if the view already exists.

- Query a view.

  Query **myview**.

  ```
  gaussdb=# SELECT * FROM myview;
   c1
  ----
  (0 rows)
  ```

- Query views of the current user.

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

- Query all views.

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

- Query details about a specified view.

  View details about **myview**.

  ```
  gaussdb=# \d+ myview
                  View "public.myview"
   Column |  Type   | Modifiers | Storage | Description
  --------+---------+-----------+---------+-------------
   c1     | integer |           | plain   |
  View definition:
   SELECT  *
     FROM mytable
    WHERE trunc(mytable.c1::double precision) > 1000::double precision;
  ```

- Drop a view.

  Drop **myview**.

  ```
  gaussdb=# DROP VIEW myview;
  DROP VIEW
  ```

3.2.6.5 Managing Sequences

Context

A sequence is a database object that generates unique integers.

If **NO CYCLE** is specified, sequence numbers are integers that automatically increase according to a certain rule. Sequences are unique because they increase automatically. This is why they are often used as primary keys.

You can use a sequence to make a column a unique identifier in either of the following ways:

- Declare the column type as a sequence integer. The database automatically creates a sequence for the column in the background.
- Use the CREATE SEQUENCE statement to create a custom sequence and reference nextval('sequence_name') in the default value of the column.

Precautions

In non-multi-primary mode, the generation of new sequence values is managed by GTM. By default, each request for a sequence value requires a separate call to GTM. GTM then increments the current value by the defined step size and returns

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the new value to the caller. As the sole global node, GTM can become a performance bottleneck. Therefore, for operations that require frequent, high-volume sequence generation (such as data import using the Bulkload tool), you are advised not to generate sequence values by default. For example, in the following scenario, the execution efficiency of the INSERT INTO SELECT FROM statement is significantly lower than expected.

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

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

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

NOTE

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

To generate the default sequence value using BulkLoad, set sufficient cache for **newSeq1** and do not set **Maxvalue** or **Minvalue**. The database will push down the calling of nextval('sequence_name') to DNs to improve performance. Currently, the concurrent connection requests that can be processed by the GTM are limited. If there are too many DNs, a large number of concurrent connection requests will be sent to the GTM. In this case, you need to limit the bulk loading concurrency, so that DNs do not fully occupy GTM connections. If the target table is a replication table (DISTRIBUTE BY REPLICATION), pushdown cannot be performed. When the data volume is large, the performance is affected and the space may bloat sharply. After the import is complete, you need to run the **vacuum full** command to restore the data. The best way is not to use BulkLoad to generate the default sequence value.

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

Procedure

Declare the column type as sequence integer to define the identifier column. For example:

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CREATE TABLE
```

Create a sequence and use the nextval('sequence_name') function to specify the default value of a column. In non-multi-primary mode, the operation is more flexible as you can define a cache and request multiple sequence values at a time to reduce interactions with GTM, thereby improving performance.

1. Create a sequence.
   ```
   gaussdb=# CREATE SEQUENCE seq1 CACHE 100;
   ```
   If the following information is displayed, the CREATE operation is successful:
   ```
   CREATE SEQUENCE
   ```
2. Set the created sequence as the default value of a column. Then, the column has a unique identifier.
   ```
   gaussdb=# CREATE TABLE T2
   (
       id   int NOT NULL DEFAULT nextval('seq1'),
       name text
   );
   ```
   If the following information is displayed, the default value has been specified:
   ```
   CREATE TABLE
   ```
3. Join a sequence with a column.

   Join a sequence with a specified column included in a table. In this way, the sequence will be dropped when you drop its joined column or the table where the column belongs to.
   ```
   gaussdb=# ALTER SEQUENCE seq1 OWNED BY T2.id;
   ```
   If the following information is displayed, the operation is successful:
   ```
   ALTER SEQUENCE
   ```

NOTE

The preceding methods are similar, except that the second method specifies cache for the sequence. However, once cache is defined, the sequence values may be discontinuous (for example, 1, 4, and 5 are generated) and the sequence cannot be preserved. If a dependent column is specified for a sequence, the sequence will be deleted when the dependent column is deleted. A sequence shared by multiple columns is not forbidden in a database, but you are advised not to do that.

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

3.2.6.6 Managing Scheduled Jobs

Context

When customers execute time-consuming jobs (such as statistics summarization or data synchronization from other databases) during daytime, these jobs may impact the performance of regular operations. As a result, customers often choose to execute such jobs at night, which inadvertently increases their workload. To address this issue, GaussDB supports the scheduled job feature in ORA-compatible databases, allowing customers to create scheduled jobs. These jobs are automatically triggered at the scheduled time, thereby reducing customers' O&M workload.

Specifically, GaussDB uses APIs from the advanced packages **DBE_SCHEDULER** and **DBE_TASK** to create scheduled jobs, run jobs automatically, delete jobs, and

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

modify job attributes (such as the job ID, enabled/disabled status, trigger time, trigger interval, and content). The DBE_SCHEDULER API is recommended to ensure high availability and reliability and support more flexible job scheduling. For detailed API descriptions and migration guidance, refer to "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_SCHEDULER" in *Reference*.

Precautions

In non-multi-primary mode:

- The **next_run_date** column in the PG_JOB system catalog indicates the actual start time of a scheduled job, not the expected start time of a scheduled job.
- The start time of the current scheduled job depends on the actual execution time of the previous job. If **freq** is not specified, the execution time has a short deviation.

```
gaussdb=# SELECT start_date, next_run_date, interval FROM pg_job WHERE job_name = 'job1';
        start_date         |       next_run_date         |      interval
---------------------------+-----------------------------+--------------------
 2024-12-03 15:24:11.94422 | 2024-12-03 15:26:12.197623 | interval'1 minute'
(1 row)
```

Managing Scheduled Jobs

Step 1 Create a test table.

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

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

```
CREATE TABLE
```

Step 2 Create a user-defined stored procedure.

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

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

```
CREATE PROCEDURE
```

Step 3 Create a job.

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

```
gaussdb=# CALL dbe_task.submit('CALL public.prc_job_1(); ', sysdate, 'interval ''1 minute'''', :a);
id
-----
1
(1 row)
```

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

```
gaussdb=# CALL dbe_task.id_submit(1,'CALL public.prc_job_1(); ', sysdate, 'interval ''1 minute'''');
id_submit
----------


(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

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

Step 5 Stop a job.

```
gaussdb=# CALL dbe_task.finish(1,true);
finish
--------

(1 row)
```

Step 6 Start a job.

```
gaussdb=# CALL dbe_task.finish(1,false);
finish
--------

(1 row)
```

Step 7 Modify job attributes.

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

  ```
  -- Modify next_time of Job1 to execute the job one hour later.
  gaussdb=# CALL dbe_task.next_time(1, sysdate+1.0/24);
  next_time
  -----------

  (1 row)
  ```

- Modify the **interval** parameter of a job.

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

  (1 row)
  ```

- Modify the **what** parameter of a job.

  ```
  -- Set what to INSERT INTO public.test VALUES(333, sysdate+5); for Job1.
  gaussdb=# CALL dbe_task.content(1,'INSERT INTO public.test VALUES(333, sysdate+5);');
  content
  ------

  (1 row)
  ```

- Modify **next_date**, **interval**, and **what** parameters of a job.

  ```
  gaussdb=# CALL dbe_task.update(1, 'CALL public.prc_job_1();', sysdate, 'interval ''1 minute''');
  update
  --------

  (1 row)
  ```

Step 8 View the job execution status.

If a scheduled job fails to execute automatically (that is, **job_status** is **'f'**), contact an administrator to retrieve failure details from the run logs stored in **gs_log**.

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

```
LOG:  Execute Job Detail:
      job_id: 1
      what: call public.test();
      start_date: 2017-07-19 23:30:47.401818
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
job_status: failed
detail error msg: relation "test" does not exist
end_date: 2017-07-19 23:30:47.401818
next_run_date: 2017-07-19 23:30:56.855827
```

Step 9 Delete a job.

```
gaussdb=# CALL dbe_task.cancel(1);
cancel
--------


(1 row)
```

----End

Job Permission Control

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

Job Concurrency Management

In non-multi-primary mode, you can set the GUC parameter **job_queue_processes** to control the number of jobs running simultaneously.

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

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

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

To enhance real-time performance for scheduled jobs, the system has implemented caching for all jobs. If you encounter the following error during job execution, it may be due to simultaneous updates by the worker thread and

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

background thread of the scheduled jobs. In such cases, you can resolve the issue by retrying the scheduled job APIs.

```
ERROR:  tuple concurrently updated
```

3.3 Managing Permissions

You can use Data Control Language (DCL) statements to set or modify the permissions of database users or roles.

Context

Upon creation, a database object is owned by the user who created it. By default, separation of duties is disabled after database installation, meaning system administrators possess the same permissions as the object owner. Consequently, only the object owner and system administrators can query, modify, or delete the object, and grant the object's permissions to other users via GRANT by default.

To allow other users to use the object, you must grant the necessary permissions to the target users or their roles.

Precautions

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

Managing Object Permissions

Step 1 Log in to the database as the administrative user **root**.

Step 2 Create **user1** and **user2**.

```
gaussdb=# CREATE USER user1 password '********';
gaussdb=# CREATE USER user2 password '********';
```

Step 3 Log in to the database as **user2**.

Step 4 Create the user table **tbl**, grant the USAGE permission of **user2** to **user1**, and grant the SELECT permission on the **tbl** table to **user1**.

```
gaussdb=> CREATE TABLE user2.tbl(id int, name varchar(5));
CREATE TABLE
gaussdb=> GRANT USAGE ON SCHEMA user2 TO user1;
GRANT
gaussdb=> GRANT SELECT ON TABLE user2.tbl TO user1;
GRANT
```

Step 5 Log in to the database as **user1**.

Step 6 Query the **tbl** table and insert data as **user1**. The query succeeds, but the insertion fails due to insufficient permissions.

```
gaussdb=> SELECT * FROM user2.tbl;
 id | name
----+------
(0 rows)

gaussdb=> INSERT INTO user2.tbl VALUES (1,'TOM');
ERROR:  The operation permission on relation tbl is denied.
```

Step 7 Log in to the database as **user2**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 8 Revoke the SELECT permission on **tbl** from **user1** and grant the INSERT permission on **tbl** to **user1**.

```
gaussdb=> REVOKE SELECT ON TABLE user2.tbl FROM user1;
REVOKE
gaussdb=> GRANT INSERT ON TABLE user2.tbl TO user1;
GRANT
```

Step 9 Log in to the database as **user1**.

Step 10 Query the **tbl** table and insert data. The query fails due to insufficient permissions, while the insertion succeeds.

```
gaussdb=> SELECT * FROM user2.tbl;
ERROR:  The operation permission on relation tbl is denied.
gaussdb=> INSERT INTO user2.tbl VALUES (1,'TOM');
INSERT 0 1
```

Step 11 Log in to the database as the administrative user **root**.

Step 12 Clean up the environment.

```
gaussdb=# DROP USER user1, user2 CASCADE;
DROP ROLE
```

----End

Managing Users and Roles

Step 1 Log in to the database as the administrative user **root**.

Step 2 Create **role1** and **role2**.

```
gaussdb=# CREATE ROLE role1 WITH password 'xxxxxxxx';
CREATE ROLE
gaussdb=# CREATE ROLE role2 WITH password 'xxxxxxxx';
CREATE ROLE
```

Step 3 Modify **role1** to include the CREATEDB and LOGIN attributes. Retain **role2** with its default permissions.

```
gaussdb=# ALTER ROLE role1 WITH CREATEDB;
ALTER ROLE
gaussdb=#  ALTER ROLE role1 WITH LOGIN;
ALTER ROLE
```

Step 4 Obtain role information from the pg_roles system view.

```
gaussdb=# SELECT oid, rolname, rolcreatedb, rolcanlogin FROM pg_roles WHERE rolname='role1' or
rolname='role2';
  oid  | rolname | rolcreatedb | rolcanlogin
-------+---------+-------------+-------------
 16750 | role1   | t           | t
 16778 | role2   | f           | f
(2 rows)
```

Step 5 Log in to the database as a user with **role1**.

Step 6 Run the following command to create **test_db**. If "CREATE DATABASE" is displayed, the CREATE operation is successful.

```
gaussdb=# CREATE DATABASE test_db;
CREATE PLUGGABLE DATABASE
```

Step 7 Grant the permissions of **role1** to **role2**. Then **role2** inherits the object permissions of **role1**.

```
gaussdb=# GRANT role1 TO role2;
GRANT ROLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 8 Obtain role member relationships from the **pg_auth_members** system catalog.

```
gaussdb=# SELECT * FROM pg_auth_members;
 roleid | member | grantor | admin_option
--------+--------+---------+--------------
  16750 |  16778 |   16750 | f
(1 row)
```

Step 9 Log in to the database as the administrative user **root**.

Step 10 Close and drop **test_db**. Then drop **role1** and **role2**.

```
gaussdb=#  ALTER DATABASE test_db CLOSE;
ALTER PLUGGABLE DATABASE
gaussdb=#  DROP DATABASE test_db;
DROP PLUGGABLE DATABASE
gaussdb=# DROP ROLE role1,role2;
DROP ROLE
```

----End

3.4 Using Stored Procedures

Stored Procedure

In GaussDB, business rules and service logic can be stored as procedures, known as stored procedures.

A stored procedure is a combination of SQL and PL/SQL. With stored procedures, the code for executing business rules can be migrated from the application layer to the database layer for reuse. This means that the code can be called by multiple procedures after being stored just once.

For details, see "Stored Procedure" in *Reference*.

Package

A package is an object-oriented construct that encapsulates PL/SQL program elements, such as stored procedures, functions, variables, constants, and cursors. It provides a unified framework for creating, deleting, and modifying functions therein.

A package comprises two parts: package specification and package body. The declarations in the specification are accessible to external functions and anonymous blocks, whereas declarations in the body can only be accessed by functions or stored procedures within the body.

PL/SQL Identifier

Each PL/SQL subprogram, package, parameter, variable, exception, or cursor has a name, which is known as a PL/SQL identifier.

Identifiers must comply with the following naming conventions:

- Contain lowercase letters (a–z), uppercase letters (A–Z), underscores (_), digits (0–9), and dollar signs ($).
- Start with a letter or an underscore.
- Be 1 to 63 characters long.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PL/SQL Data Type

A data type defines a set of values and the operations that can be performed on those values. GaussDB is composed of a collection of tables, where each table is defined by its columns. Each column is assigned a specific data type, and GaussDB provides corresponding functions to manipulate the column's content based on its data type. For example, GaussDB can perform addition, subtraction, multiplication, and division on numeric data.

In PL/SQL, SUBTYPE allows users to create subtypes based on any basic or user-define type.

Basic PL/SQL Structure

A PL/SQL block can contain subblocks, and subblocks can be placed in any section of a PL/SQL block. A PL/SQL block is structured as follows:

- **DECLARE**: declares the variables, types, cursors, and local stored procedures and functions used in the PL/SQL block.
  ```
  DECLARE
  ```

  NOTE

  The **DECLARE** section is optional if no variables are declared.

  - For anonymous blocks, the DECLARE keyword may be omitted when no variable declaration is present.
  - For stored procedures, the AS keyword is equivalent to DECLARE when the DECLARE keyword is absent. The AS keyword must be retained even if no variables are declared.
- **BEGIN**: contains procedures and SQL statements. As the program's main section, it is required.
  ```
  BEGIN
  ```
- **EXCEPTION**: defines error handling logic. This section is optional.
  ```
  EXCEPTION
  ```
- **END**: marks the conclusion of the block. This section is required.
  ```
  END;
  /
  ```

  NOTICE

  - Consecutive tab characters are prohibited in PL/SQL blocks, as they may cause exceptions when gsql tool is executed with the **-r** parameter.
  - Only the top-level anonymous block requires the / terminator.

PL/SQL blocks are classified into the following types:

- Anonymous block: An anonymous block is executed only once. It is written during execution, and its result is not stored in the database.
- Program: A program refers to a stored procedure, function, operator, or advanced package in the database. They can be called by other programs once the database is properly set up.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PL/SQL Basic Statement

- Variable declaration

  Variables are declared in the **DECLARE** section, and each variable requires a specific type. You can assign an initial value to a variable in its declaration.

```
DECLARE
id      INTEGER;  -- Declare the variable.
name    VARCHAR(20);  -- Declare the variable.
age     INT := 20;  -- Declare the variable and assign a value to it.
weight  NUMBER(5,2) := 60.5;  -- Declare the variable and assign a value to it.
address VARCHAR(20) := 'xian';  -- Declare the variable and assign a value to it.
BEGIN
END;
/
ANONYMOUS BLOCK EXECUTE
```

- Assignment statement

  In BEGIN...END blocks, values are typically assigned to variables by using = or :=.

```
DECLARE
id      INTEGER;  -- Declare the variable.
name    VARCHAR(20);  -- Declare the variable.
BEGIN
id := 1;  -- Assign a value to the variable.
name := 'xiaohua';  -- Assign a value to the variable.
END;
/
ANONYMOUS BLOCK EXECUTE
```

PL/SQL Dynamic Statement

Dynamic statements allow programs to dynamically construct and execute SQL statements or PL/SQL blocks at runtime.

```
gaussdb=# CREATE SCHEMA use_procedure;
CREATE SCHEMA
gaussdb=# CREATE TABLE use_procedure.tb1 (a int);
CREATE TABLE
gaussdb=# INSERT INTO use_procedure.tb1 VALUES (generate_series(1, 10));
INSERT 0 10
gaussdb=# DECLARE
counts int;
BEGIN
EXECUTE IMMEDIATE 'select count(*) from use_procedure.tb1' INTO counts;
dbe_output.print_line(counts);
END;
/
10
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP SCHEMA use_procedure CASCADE;
NOTICE:  drop cascades to table use_procedure.tb1
DROP SCHEMA
```

PL/SQL Control Statement

The primary purpose of conditional statements is to evaluate whether parameters or statements meet the given conditions and to execute the corresponding action based on the evaluation result.

```
gaussdb=# DECLARE
count int := 0;
BEGIN
LOOP
  IF count  > 10 THEN
    EXIT;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  ELSE
    count := count + 1;
  END IF;
END LOOP;
END;
/
ANONYMOUS BLOCK EXECUTE
```

PL/SQL Transaction

Calling a stored procedure automatically starts a transaction. At the end of the call, the transaction is automatically committed, or rolled back in case of an exception. In addition to the system's automatic transaction control, you can leverage the COMMIT and ROLLBACK statements to manage transactions within stored procedures. Calling COMMIT or ROLLBACK in a stored procedure will commit or roll back the current transaction and automatically initiate a new one. All subsequent operations will then execute within this new transaction.

```
gaussdb=# CREATE SCHEMA use_procedure;
CREATE SCHEMA
gaussdb=# CREATE TABLE use_procedure.tb1 (a int);
CREATE TABLE
gaussdb=# DECLARE
BEGIN
FOR i IN 1..10 LOOP
  INSERT INTO use_procedure.tb1 VALUES (i);
  IF i % 2 = 0 THEN
    COMMIT;
  ELSE
    ROLLBACK;
  END IF;
END LOOP;
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP SCHEMA use_procedure CASCADE;
NOTICE:  drop cascades to table use_procedure.tb1
DROP SCHEMA
```

PL/SQL Cursor

To process SQL statements, the stored procedure process assigns a memory area to store contextual relationships. Cursors are handles or pointers to the context area. With cursors, stored procedures can control alterations in the context area.

```
gaussdb=# CREATE SCHEMA use_procedure;
CREATE SCHEMA
gaussdb=# CREATE TABLE use_procedure.tb1 (a int);
CREATE TABLE
gaussdb=# INSERT INTO use_procedure.tb1 VALUES (generate_series(1, 10));
INSERT 0 10
gaussdb=# DECLARE
CURSOR my_cur is SELECT a FROM use_procedure.tb1;
tb_a int;
BEGIN
  OPEN my_cur;
  LOOP
    FETCH my_cur into tb_a;
    EXIT WHEN my_cur%NOTFOUND;
  END LOOP;
  CLOSE my_cur;
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP SCHEMA use_procedure CASCADE;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  drop cascades to table use_procedure.tb1
DROP SCHEMA
```

Managing Stored Procedures

- Creating a stored procedure

  ```
  gaussdb=# CREATE OR REPLACE PROCEDURE proc1() AS
  DECLARE
  id INT;
  name VARCHAR(20);
  BEGIN
  id := 1;
  name := 'xian';
  END;
  /
  CREATE PROCEDURE
  ```

- Modifying a stored procedure

  ```
  -- Change the stored procedure name.
  gaussdb=# ALTER PROCEDURE proc1() RENAME TO proc2;
  ALTER FUNCTION
  -- Recompile a stored procedure by name.
  gaussdb=# ALTER PROCEDURE proc2() COMPILE;
  ALTER PROCEDURE
  ```

- Using a stored procedure

  ```
  -- Directly call it.
  gaussdb=# CALL proc2();
   proc2
  -------

  (1 row)

  -- Call it with an anonymous block.
  gaussdb=# BEGIN
  proc2();
  END;
  /
  ANONYMOUS BLOCK EXECUTE
  ```

- Dropping a stored procedure

  ```
  gaussdb=# DROP PROCEDURE proc2;
  DROP PROCEDURE
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4 Development and Design Proposal

4.1 Overview

Overview

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

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

Design specifications in terms of database and performance

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

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

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

Terms

This document uses the following terms for description:

- **Specification**: database specification that must be followed during programming and design. Otherwise, the database will report an error.
- **Rule**: a convention that must be followed during programming and design.
- **Recommendation**: a convention that must be considered during programming and design.
- **Description**: necessary clarification for a rule or recommendation.
- **Example**: case demonstrating correct or incorrect implementation of a rule or recommendation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Applicability

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

4.2 Database Design Specifications

4.2.1 General Specifications

- [Rule] Valuable database features

Table 4-1 Recommended valuable database features

Category	Feature	Description
Table type	Distributed table	Shards data.
	Partitioned table	Partitions data.
Storage engine	Row store	Stores tables by row, which is recommended for point queries and scenarios involving frequent insert, delete, and update operations.
Transaction	Transaction block	Explicitly starts a transaction.
	Single-statement transaction	Does not explicitly start a transaction. A single statement is a transaction.
	Distributed transaction (weak consistency)	Supports the GTM-free mode, which does not allow strong-consistency reads of distributed transactions but can ensure strong consistency in sharding scenarios.
Deployment	One primary and multiple standby DNs	Employs DN sharding (one primary DN + multiple standby DNs) to ensure data backup and DR.
Security	Transparent data encryption (TDE)	Encrypts stored data at the data table level. Upper-layer services are unaware of the encryption.
Data type	Integer	TINYINT, SMALLINT, INTEGER, and BIGINT.
	Arbitrary-precision	NUMERIC and DECIMAL.
	Floating-point	REAL/FLOAT4, DOUBLE PRECISION/FLOAT8, and FLOAT.
	Boolean	Boolean.
	Fixed-length character	CHAR(n).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Feature	Description
	Variable-length character	VARCHAR(n), NVARCHAR2(n), and TEXT.
	Time	DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, SMALLDATETIME, INTERVAL, RELTIME, and DATEA (effective only in ORA-compatible mode).
	Binary	BYTEA (variable-length binary type).
	Bit string	BIT(n) and VARBIT(n).
Function	Character processing function	Processes data of the character type.
	Binary string function	Processes data of the binary string type.
	Numeric operation function	Processes data of a numeric type.
	Time and date processing function	Processes data of a time and date type.
Index	Primary key or unique index	Supports primary keys and unique indexes with a single or multiple columns.
	B-tree index	Serves as an index type.
Scale-out	Online scale-out	Ensures smooth transition of user services during node scale-out and data redistribution.

- [Rule] Recommended best practices for database usage

Table 4-2 Recommended best practices for database usage

No.	Item	Recommended Value
1	Optimal number of shards in a cluster (number of primary DNs)	< 256.
2	Optimal number of long connections in a cluster	Refer to the default configuration of the GUC parameter **max_connections** under relevant hardware specifications.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

No.	Item	Recommended Value
3	Maximum data volume on a single physical node	16 TB (to be finalized based on backup and restoration specifications).
4	Number of active databases in a cluster	1.
5	Total number of cluster tables and indexes	10,000 (recommended: no more than 200 tables per schema). **NOTE** The value is only a recommended value. You need to plan the value based on the disk capacity and customer services. Theoretically, the maximum value is 2^{32}.
6	Optimal number of columns in a single table	< 50.
7	Optimal number of indexes in a table	< 5.
8	Optimal number of composite indexes in a table	< 3.
9	Optimal number of columns in a single composite index	< 5.
10	Optimal width of a single row	< 2,000.
11	Optimal capacity of a single column	• In non-multi-primary mode: < 10 MB.
12	Optimal SQL statement length	< 5,000.
13	Percentage of available disk space	• In non-multi-primary mode: < 50%.

4.2.2 Deployment Specifications

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

GaussDB provides various methods for modifying GUC parameters, making it convenient to configure databases, users, and sessions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Contact an administrator for GUC parameter settings.

4.2.3 Database Object Naming Specifications

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

  You can run the following command to query database keywords, or refer to "SQL Reference > Keywords" in *Reference*.

  ```
  SELECT * FROM pg_get_keywords();
  ```

- [Specification] The database object name contains a maximum of 63 bytes.

  Table 4-3 Length limits for database object names

Object Type	Length Limit (bytes)	Remarks
Database name	≤ 63	-
Table name	≤ 63	• In non-multi-primary mode, when copying or importing GDS data into a table with a global secondary index (GSI), ensure that the length of the table name does not exceed 38 bytes. Otherwise, the temporary table name may be truncated, interrupting the data import process and resulting in issues such as unexpected characters.
Table object	≤ 63	If the length of a table name exceeds this value, the kernel truncates the table name. As a result, the actual name is inconsistent with the configured value. In addition, characters may be truncated in different character sets and unexpected characters may appear.
Temporary table not used by services	≤ 63	For example, an intermediate table used by operation personnel for temporary backup or data collection. Naming rule: **tmp**_*Table name abbreviation_Creator account abbreviation_Creation date*, for example, **tmp_user_ytw_160505**.
Table column	≤ 63	Meaningful English words or abbreviations are used to name the column. For example, a column of the bool type is named in the format of **is**_*Description*. A column whose member status is **enabled** in the member table is named **is_enabled**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Object Type	Length Limit (bytes)	Remarks
View name and column name	≤ 63	-
Function name and parameter name	≤ 63	-

- [Rule] Do not use a string enclosed in double quotation marks ("") to define the database object name, unless you must specify its capitalization.
  - By default, GaussDB treats object names in SQL statements as case-insensitive. If tables like **t_Table** and **t_table** co-exist in the same database, you must use double quotation marks (""). If there are no such tables, avoid using double quotation marks.
  - Case sensitivity of database object names makes fault locating difficult.
- [Rule] Identifiers must meet the following requirements:
  - An identifier starts with a letter or underscore (_) and can contain letters, digits, underscores (_), dollar signs ($), and number signs (#).
  - If an identifier is enclosed by backquotes (``) in MySQL-compatible mode or by double quotation marks (""), it can be a combination of any valid characters, for example, **"123gs_column"**.
  - Identifiers are case-insensitive. Only those enclosed in double quotation marks, or in backquotes (``) in MySQL-compatible mode, are considered case-sensitive.
- [Rule] The gsql shortcut commands (except **\sf**) cannot be used to query the object names enclosed in backquotes.
- [Recommendation] Use the same naming style for database objects.
  - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.
  - You are advised to use multiple words separated with underscores (_).
  - You are advised to use intelligible names and common acronyms or abbreviations for database objects. For example, you can use English words or Chinese pinyin indicating actual business terms. The naming format should be consistent within a cluster.
  - A variable name must be descriptive and meaningful. It must have a prefix indicating its type.
- [Recommendation] The name of a table object should indicate its main characteristics, for example, whether it is an ordinary, temporary, or unlogged table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - 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*.
  - Do not create database objects whose names start with **redis**_.
  - Do not create database objects named **pgxc_redistb**.
  - Do not create database objects whose names start with **mlog**_ or **matviewmap**_.
  - Do not create database objects whose names start with **gs_role**_.

4.2.4 Database and Schema Design Specifications

- [Description] When you create a database, exercise caution when setting the **ENCODING** and **DBCOMPATIBILITY** configuration items. GaussDB supports five compatibility modes: TD (Teradata), ORA (Oracle), MySQL, PG (Postgres), and M (M compatibility). The syntax behavior varies depending on the compatibility mode. By default, the MySQL-compatible mode is used.
- [Description] By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when deleting a permission.
- [Description] To let a user access an object in a schema, grant the usage permission and the permissions on the object to the user, unless the user has the SYSADMIN permission or is the schema owner.
- [Description] To let a user create an object in the schema, grant the CREATE permission on the schema to the user.
- [Description] By default, a schema owner has all permissions on all objects in the schema, including the deletion permission. Exercise caution when deleting a permission.
- [Specification] The database name must be specified when the JDBC client is used to connect to the database. The format is as follows:

```
jdbc:gaussdb://ip:port/database_name
```

  NOTICE

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

- [Specification] Case-insensitive sorting is supported only in MySQL-compatible (**sql_compatibility = 'MYSQL'**) and M-compatible (**sql_compatibility = 'M'**) modes.
- [Rule] Before using services, a system administrator must create databases, schemas, and users for services, and then grant object permissions to corresponding users.
- [Rule] Create a service database before using a service. Do not use the Postgres database created by default after the database is installed to store service data. You are advised to create your own database based on service requirements.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- [Rule] When creating a database, you must specify the character set as **UTF8** and maintain consistency with the client-side encoding.

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

  Inconsistent encoding modes between the client and the database necessitate transcoding, which affects performance. In addition, kernel optimization for the same encoding cannot be triggered, reducing query efficiency.

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

  – Set client connection parameters. Take JDBC as an example. Add the parameters **characterEncoding** and **allowEncodingChanges** to the URL.
  ```
  jdbc:gaussdb://ip:port/database_name?characterEncoding=utf8&allowEncodingChanges=true
  ```
  – Modify a database GUC parameter.
  ```
  SET client_encoding = 'UTF8';
  ```

  Set the database encoding when executing the **CREATE DATABASE**.
  ```
  CREATE DATABASE tester WITH ENCODING = 'UTF8';
  ```

  NOTICE

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

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

  Services can be isolated via databases or schemas. The key differences are:

  – Databases share few resources. Connections to and permissions on them are also isolated. However, the databases cannot access each other. The database must be specified during JDBC connection establishment. After the connection is established, the database cannot be switched.

  – Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be controlled using the GRANT and REVOKE syntax.
- [Recommendation] You are advised to specify **LC_CTYPE** and **LC_COLLATE** when creating a database. These parameters will affect the data collation sequence.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.2.5 Permission Design Specifications

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

  NOTE

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

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

  Use roles to manage permissions, that is, configure permissions for roles and grant the roles to users. Roles facilitate permission management in scenarios such as multiple users and user changes. Example:

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

- [Recommendation] When deleting a specified database, revoke the CONNECT permission of users on the database to prevent deletion failures caused by active database connections.
- [Recommendation] Avoid the risk of permission exploitation due to improper use of permissions.

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

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

4.2.6 Character Set Design Specifications

- [Specification] Currently, except MySQL- or M-compatible databases (**sql_compatibility** is set to **'MYSQL'** or **'M'**), character sets can be defined only for databases, but not for other objects such as tables or columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

In MySQL- or M-compatible databases (**sql_compatibility** is set to **'MYSQL'** or **'M'**), the character set of objects such as tables and columns must be the same as that of the database. Otherwise, an error is reported.

- [Recommendation] The database and client use the same UTF-8 character set.
  - Clients include ODBC, JDBC, and GSQL.
  - For reasons, see **Database and Schema Design Specifications**

4.2.7 Table Design Specifications

- [Rule] View nesting is not recommended.
  - If a wildcard is used when a view is written, an error occurs in the view when a column is added to or deleted from the called view.
  - View nesting may be inefficient because indexes cannot be used. You are advised to use base tables with indexes instead of performing join operations on views.
- [Recommendation] Try to avoid collation operations when defining a view.

  ORDER BY is invalid in the top-level view. If you must collate the output data, use ORDER BY in a called view.
- [Description] In non-multi-primary mode, table scan pressure is evenly distributed to each DN. Otherwise, DNs most frequently scanned will become the performance bottleneck. For example, when you use equivalent filter conditions on a fact table, the nodes are not evenly scanned.
- [Recommendation] Minimize random I/Os. Through clustering, you can sequentially store hot data, converting random I/O to sequential I/O to reduce the cost of I/O scanning.
- [Recommendation] In non-multi-primary mode, avoid shuffling data. 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 Distribution Mode

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

Table 4-4 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenario
Hash	Table data is distributed on all DNs in a cluster by hash.	Fact tables containing a large amount of data.
Replication	Full data in a table is stored on every DN in the cluster.	Dimension tables and fact tables containing a small amount of data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Distribution Mode	Description	Application Scenario
Range	Table data is mapped to specified columns based on the range and distributed to the corresponding DNs.	Users need to customize distribution rules.
List	Table data is mapped to specified columns based on specific values and distributed to corresponding DNs.	Users need to customize distribution rules.

- [Description] Evenly distribute table data on each DN to prevent data skew. If most data is stored on several DNs, the effective capacity of a cluster decreases. Select a proper distribution key to avoid data skew.
- [Rule] DISTRIBUTE BY must be specified and the table distribution policy must comply with the following principles.
  - For system configuration tables, data dictionary tables, and other tables with data volumes under 20 million and infrequent inserts/updates, the replication distribution is required.

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

    NOTICE

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

  - For tables not meeting the above criteria—that is, those with large data volumes and high update frequency—data must be sharded, and the hash distribution policy should be applied. The distribution key must be one or more columns in the primary key.

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

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

The example of defining a distributed table is as follows:

```
-- Define a table with each row stored in all DNs.
CREATE TABLE warehouse_d1
(
    W_WAREHOUSE_SK           INTEGER               NOT NULL,
    W_WAREHOUSE_ID           CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT        INTEGER                       ,
    W_STREET_NUMBER          CHAR(10)                      ,
    W_STREET_NAME            VARCHAR(60)                   ,
    W_STREET_TYPE            CHAR(15)                      ,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY REPLICATION;

-- Define a hash table.
CREATE TABLE warehouse_d2
(
    W_WAREHOUSE_SK            INTEGER                NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)               NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                    ,
    W_WAREHOUSE_SQ_FT         INTEGER                        ,
    W_STREET_NUMBER           CHAR(10)                       ,
    W_STREET_NAME             VARCHAR(60)                    ,
    W_STREET_TYPE             CHAR(15)                       ,
    W_SUITE_NUMBER            CHAR(10)                       ,
    W_CITY                    VARCHAR(60)                    ,
    W_COUNTY                  VARCHAR(30)                    ,
    W_STATE                   CHAR(2)                        ,
    W_ZIP                     CHAR(10)                       ,
    W_COUNTRY                 VARCHAR(20)                    ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
)DISTRIBUTE BY HASH(W_WAREHOUSE_SK);

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

-- Define a table using LIST distribution.
CREATE TABLE warehouse_d4
(
W_WAREHOUSE_SK            INTEGER                NOT NULL,
W_WAREHOUSE_ID            CHAR(16)               NOT NULL,
W_WAREHOUSE_NAME          VARCHAR(20)                    ,
W_WAREHOUSE_SQ_FT         INTEGER                        ,
W_STREET_NUMBER           CHAR(10)                       ,
W_STREET_NAME             VARCHAR(60)                    ,
W_STREET_TYPE             CHAR(15)                       ,
W_SUITE_NUMBER            CHAR(10)                       ,
W_CITY                    VARCHAR(60)                    ,
W_COUNTY                  VARCHAR(30)                    ,
W_STATE                   CHAR(2)                        ,
W_ZIP                     CHAR(10)                       ,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
W_COUNTRY                VARCHAR(20)                ,
W_GMT_OFFSET             DECIMAL(5,2)
)DISTRIBUTE BY LIST(W_COUNTRY)
(
   SLICE s1 VALUES ('USA') DATANODE dn1,
   SLICE s2 VALUES ('CANADA') DATANODE dn2,
   SLICE s3 VALUES ('UK') DATANODE dn3,
   SLICE s4 VALUES (DEFAULT) DATANODE dn4
);
```

Selecting a Distribution Key

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

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

Table 4-5 Common distribution keys and effects

Distribution Key Value	Distribution Effect
User ID. Many users exist in an application.	Good

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Distribution Key Value	Distribution Effect
Status code. Only several status codes are available.	Poor
Date when a project is created. The value is rounded off to the latest time segment (for example, day, hour, or minute).	Poor
Device ID. Each device accesses data at a relatively similar interval.	Good
Device ID. Many devices are traced. However, one particular device has been used significantly more frequently than all others so far.	Poor

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

Selecting a Partitioning Mode

Table 4-6 Table partitioning modes and scenarios

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

Comply with the following rules when partitioning a table with a large amount of data:

- [Description] Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.
- [Recommendation] Create partitions on columns that indicate certain ranges, such as dates and regions.
- [Recommendation] A partition name should show the data characteristics of a partition. For example, its format can be *Keyword+Range* characteristics.
- [Recommendation] Define the upper boundary value of the partition as **MAXVALUE** to prevent data overflow.
- [Recommendation] In non-multi-primary mode, avoid defining identical or similar partitioning and distribution rules. If such rules are applied, partitions on certain DNs may fail to map to data.

The example of defining a partitioned table is as follows:

```
-- Set the DATE type format.
SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a range partitioned table.
CREATE TABLE staffS_p1
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
(
   PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
   PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
   PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);

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

-- Create a hash partitioned table.
CREATE TABLE test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);
```

4.2.8 Column Design Specifications

- [Rule] Use recommended data types for column design.

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

  Table 4-7 Best practices for database data types

Data Type	Description	Recommended or Not
UUID	Different clusters may generate the same UUID.	Prohibited
Serial integer	Auto-increment column, including SMALLSERIAL, SERIAL, and BIGSERIAL.	Prohibited
Integer	TINYINT, SMALLINT, INTEGER, and BIGINT.	Recommended

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Type	Description	Recommended or Not
Arbitrary-precision	NUMERIC and DECIMAL.	Recommended
Floating-point	REAL/FLOAT4, DOUBLE PRECISION/ FLOAT8, and FLOAT.	Recommended
Boolean	Boolean.	Recommended
Fixed-length character	CHAR(n).	Recommended
Variable-length character	VARCHAR(n) and NVARCHAR2(n).	Recommended
	TEXT and CLOB (character large object).	Not recommended
Time	DATE, TIME, TIMESTAMP, SMALLDATETIME, INTERVAL, RELTIME, and DATEA (effective only in ORA-compatible mode).	Recommended
	TIMETZ and TIMESTAMPTZ.	Not recommended
Binary	BYTEA (variable-length binary type).	Recommended
	BLOB (binary large object) and RAW (variable-length hexadecimal string).	Not recommended
Bit string	BIT(n) and VARBIT(n).	Recommended
Special character	NAME and "CHAR" are usually used within the database system.	Not recommended
JSON	The JSON type currently does not support operators.	Not recommended
User-defined	It is used to define enumerated types.	Not recommended
HLL	You are advised to use the HLL functions to reduce the impact on performance.	Not recommended
Currency	The MONEY type stores a currency amount with fixed fractional precision.	Not recommended
Geometric	POINT, LSEG, BOX, PATH, POLYGON, and CIRCLE.	Not recommended

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Type	Description	Recommended or Not
Network address	Stores IPv4, IPv6, and MAC addresses.	Not recommended

- [Rule] Use the most specific numeric data types. If all of the following numeric types provide the required service precision, they are recommended in descending order of priority: integer, floating point, and NUMERIC.
- [Rule] Properly set the data type of a numeric column based on the value range, and use the NUMERIC or DECIMAL type as less as possible.
  - NUMERIC and DECIMAL are equivalent. NUMERIC or DECIMAL data operations consume great CPU resources.
  - When **sql_compatibility** is set to **'MYSQL'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the value of TINYINT ranges from –128 to +127.
  - The UNSIGNED type is available only when **sql_compatibility** is **'MYSQL'**.

  Table 4-8 Storage space and value range for numeric data types in non-multi-primary mode

Type	Storage Space	Minimum Value	Maximum Value
TINYINT [UNSIGNED]	1	0	255
SMALLINT [UNSIGNED]	2	Signed: **-32768**; Unsigned: **0**.	Signed: **32767**; Unsigned: **65535**.
INTEGER [UNSIGNED]	4	Signed: **-2,147,483,648**; Unsigned: **0**.	Signed: **2,147,483,647**; Unsigned: **4,294,967,295**.
BIGINT [UNSIGNED]	8	Signed: **-9,223,372,036,854,775,808**; Unsigned: **0**.	Signed: **9,223,372,036,854,775,807**; Unsigned: **18,446,744,073,709,551,615**.
REAL/FLOAT4	4	6 decimal digits	
DOUBLE PRECISION/ FLOAT8	8	15 decimal digits	

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For a typical fixed-length column, for example, **gender**, you can enter only **f** or **m** that occupies a byte. For such columns, it is advisable to use a fixed-length data type—for example, CHAR(*n*).

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

The reasons are as follows:

- For fixed-length columns, the input data that is shorter than the fixed length will be padded with space characters and then be saved to the database. This wastes the storage space in the database.
- For fixed-length character types, the entire table needs to be scanned and rewritten if the length needs to be extended later. This causes high performance overhead and affects online services.
- For a variable-length column with a fixed length, the system checks whether the length exceeds the limit each time upon data insertion. This causes performance overhead.

- [Rule] Do not store data of the numeric type in columns of the character type.

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

  If calculation or comparison (for example, adding a filter condition) with data of the time or date type is performed on data stored in columns of the character type, unnecessary overhead will be caused by data type conversion, and the column indexes may become invalid, affecting query performance.
- [Rule] Add the NOT NULL constraint for columns that do not have NULL values.

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

  If the column types are inconsistent during a join operation, overhead will be caused by data type conversion.
- [Rule] Use no more than eight large columns, such as VARCHAR(1000) or VARCHAR(4000).
- [Recommendation] When defining a column, you are advised to create a comment for the column to facilitate subsequent maintenance.

  For details about the descriptions, value ranges, and usage of different types of columns, see "SQL Reference > Data Types" in *Reference*.
- [Recommendation] In tables that are logically related, columns having the same meaning should use the same data type.
- [Recommendation] Add NOT NULL constraints to columns that are used for WHERE filtering and join operations.

  In certain scenarios, the optimizer may specially optimize NOT NULL columns to greatly improve query performance.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- [Recommendation] Do not reserve columns for a table. In most cases, you can quickly add or delete table columns, or change the default values of columns.

  An added column must meet the following requirements. Otherwise, the entire table is updated, leading to additional overheads and affecting online services.

  – The data type can be BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, INTERVAL, or DATEA (effective only in ORA-compatible mode).
  – The length of the default value cannot exceed 128 bytes.
  – The default value of the added column does not contain the volatile function.
  – The default value is required and cannot be **NULL**.

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

4.2.9 Index Design Specifications

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

  NOTE

  Recommended types must be used for index design.

  Table 4-9 Recommended practices for database indexes

Index Type	Description	Recommended or Not
Primary key or unique index	Single-column or multi-column primary key or unique index.	Recommended
Expression index	An index column is a function or scalar expression calculated from one or multiple columns in a table.	Restricted
B-tree index	Index building type.	Recommended for some data types

- [Rule] A composite index contains a maximum of 5 columns.
- [Rule] Composite indexes longer than 200 characters are prohibited.
- [Rule] Columns with indexes (whether single-column or composite indexes) should be defined as NOT NULL.
- [Rule] Do not create multiple unique indexes for a single table.

  Maintaining multiple unique indexes at the same time generates more overheads than maintaining a multi-column unique index. If multiple unique

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

indexes are equivalent to a multi-column unique index in service logic, a multi-column unique index shall be preferred.

- [Recommendation] The efficiency of maintaining indexes created for the same column is different. Columns of the number type are better than those of the character type and other data types. So, it is recommended that columns such as IDs for creating indexes be stored as data of the number type.
- [Recommendation] Create indexes on joined columns.

  The system supports hash joins, but nested-loop joins may be selected when rescan costs are low, such as when the inner table is small. If the **NESTLOOP JOIN** plan can be viewed by executing **EXPLAIN**, you can create indexes on joined columns to improve the efficiency of executing **NESTLOOP JOIN**.

4.2.10 Function/Stored Procedure Design Specifications

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

  Currently, the database supports three types of functions: IMMUTABLE, STABLE, and VOLATILE.

  In non-multi-primary mode, the functions are executed on DNs. In this scenario, data on all DNs is sent to one CN for computing, resulting in low query efficiency.

4.2.11 Constraint Design

DEFAULT and NULL Constraints

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

Unique Constraints

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

Primary Key Constraints

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Check Constraints

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

4.2.12 View and Joined Table Design

View Design

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

Joined Table Design

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

4.3 Database Programming Specifications

4.3.1 GUC Parameter Programming Specifications

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

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

If GUC parameters must be set in a connection, you must run the following command to clear the connection status before returning the connection to the connection pool:

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

4.3.2 Object Access Programming Specifications

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

  For details about the permission description, see **Users and Permissions** and comply with **Permission Design Specifications**.

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.3.3 WHERE

- [Rule] Do not compare the same table columns in WHERE conditions.

  For example, the following statement does not meet specifications:

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

  The following modification should be considered:

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

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

  After implicit conversion is performed in the database, the created index may fail to be used. You are advised to enable the GUC parameter **check_implicit_conversions** and disable **enable_fast_query_shipping** during development to check whether query statements contain implicit data types that may affect performance.

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

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

  Example:

  – The following codes do not meet specifications:

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

  – The following modification should be considered:

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

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

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

  Example:

  – The following codes do not meet specifications:

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

  – The following modification should be considered:

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

- [Rule] During table query, the WHERE condition must contain all partitioning keys. Otherwise, the query will be performed on multiple nodes, affecting the system concurrency and performance.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- [Rule] Do not use the comparison operator (!=) when comparing with NULL in query conditions. Instead, use IS NULL or IS NOT NULL.
- [Rule] Do not use comparison operator (!=) in the index columns of the query condition to avoid invalid indexes.
- [Rule] Do not compare the index columns of the query condition with NULL (IS NULL and IS NOT NULL instead).
- [Rule] Do not use NOT in the index columns of query conditions.
- [Rule] Do not use NOT IN in the index columns of query conditions.
- [Rule] Do not place % at the beginning of the LIKE statement for fuzzy query.

  If % is placed at the beginning of the LIKE statement, the index cannot be used and the entire table will be scanned.
- [Recommendation] Maintain a small candidate subset for the IN clause in the WHERE condition, ideally not exceeding 500.
  - During the query, the values of all the IN clauses will be compared to check whether they are equal, which increases the overhead.
  - In non-multi-primary mode, if the included values are relatively fixed, consider creating a replication table, writing candidate data into it, and then performing an INNER JOIN operation to implement the inclusion query.
- [Recommendation] If the IN clause in the WHERE condition is not a constant but a column in the table, you are advised to change it to a subquery.

  In this case, it is actually an unequal JOIN, which is executed through the nestloop plan. If the table is too large, the execution efficiency is low. You are advised to change the query to a subquery of the JOIN type. Example:
  - The following code does not meet specifications:

    ```
    SELECT col1, COALESCE(max(col2 - 1), 0)
     FROM t1, t2
     WHERE t1.col1 = ANY(VALUES(id1), (id2))
     GROUP BY col1;
    ```
  - The following modification should be considered:

    ```
    SELECT col1, COALESCE(max(tmp), 0) FROM
    (
      (
        SELECT col1, (col2-1) AS tmp
          FROM t1, t2
          WHERE t1.col1 = t2.id1 AND t1.col1 != t2.id2
      ) UNION ALL (
        SELECT col1, (col2-1) AS tmp
          FROM t1, t2
          WHERE t1.col1 = t2.id2
      )
    ) GROUP BY col1;
    ```
- [Recommendation] Use equal operators, instead of not equal operators.

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

4.3.4 SELECT

- [Rule] Do not use the wildcard character (*) in the SELECT statement.

  If the definition of a table changes due to a service or database upgrade, incompatibility with service statements may occur when you query the table

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

by using a wildcard column. Therefore, the service must specify the name of the table column to be queried and avoid using the wildcard character.

- [Rule] SELECT LIMIT statements must contain ORDER BY for query.

  GaussDB is a distributed database, meaning that table data is distributed across multiple DNs. If the SQL statement contains only LIMIT but not ORDER BY, the database sends the result (meeting the query requirements) sent by the DN with fast network transmission to the client as the final result. The network transmission efficiency may change at different time points. As a result, the returned results are inconsistent when the SQL statement is executed for multiple times.

- [Rule] Do not perform operations that may cause sorting, such as ORDER BY, DISTINCT, GROUP BY, and UNION, on large columns (such as VARCHAR(2000)).

  These operations consume a large number of CPU and memory resources, resulting in low execution efficiency.

- [Rule] Do not use the LOCK TABLE statement. Instead, use the SELECT .. FOR UPDATE statement.

  LOCK TABLE provides multiple levels of locks. However, if you do not fully understand the database principles and services, misuse of table locks may trigger deadlocks, causing the cluster to be unavailable.

- [Recommendation] Do not use subqueries in the SELECT target columns. Otherwise, the plan may fail to be pushed down to DNs for execution, affecting the execution performance.

- [Recommendation] Use UNION ALL instead of UNION. Consider data deduplication if necessary.

  UNION ALL does not deduplicate data and does not require sorting operations. Therefore, UNION ALL is faster than UNION. If deduplication is not required, UNION ALL is preferred.

- [Recommendation] Do not frequently use count() to obtain the number of rows in a large table. This operation consumes a large number of resources and affects the execution efficiency of parallel jobs.

  If you do not need the real-time row statistics, run the following statement to obtain the number of rows in the table:

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

  The number of rows recorded in pg_class is updated only after ANALYZE is executed on the table. Currently, ANALYZE is triggered in either of the following conditions:

  – The service sends an ANALYZE statement. For example:

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.3.5 INSERT

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

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

- [Rule] In non-multi-primary mode, INSERT ON DUPLICATE KEY UPDATE is prohibited on tables with multiple UNIQUE constraints.

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

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

4.3.6 UPDATE

- [Specification] In non-multi-primary mode, the GTM-free mode does not allow cross-node transactions. Therefore, when updating data in a hash distributed table, you must specify equality for distribution keys in the WHERE condition.
- [Rule] The UPDATE statement must contain the WHERE clause to avoid full table scan.
- [Rule] Do not use updated columns as the update source in multi-column UPDATE operations.

  Even when multiple columns are updated simultaneously from the same source, database behavior varies. To prevent compatibility issues, refrain from such operations at the service layer. For example:

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

  In GaussDB, this statement causes **col3** to retain the original value of **col1**, whereas in MySQL, **col3** takes the value of **col2** (because the value of **col2** is assigned to **col1**).

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

4.3.7 DELETE

- [Specification] The non-multi-primary mode does not support using LIMIT in the DELETE statement. The WHERE condition should be used to specify the target row to be deleted.
- [Specification] In non-multi-primary mode, the GTM-free mode does not allow cross-node transactions. Therefore, when deleting data from a hash

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

distributed table, you must specify equality for distribution keys in the WHERE condition.

- [Rule] The DELETE statement must contain the WHERE clause to avoid full table scan.
- [Rule] To prevent unnecessary sorting, do not use the ORDER BY or GROUP BY clause in the DELETE statement.
- [Rule] Use TRUNCATE instead of DELETE to clear a table.
  - In non-multi-primary mode, TRUNCATE creates a physical file and deletes the original file physically at the end of the transaction to free up disk space. In contrast, DELETE only marks data in the table and does not actually free up disk space until the VACUUM FULL phase.
- [Recommendation] If **DELETE** is executed on a table that has a primary key or index, the WHERE condition must be used together with the primary key or index to improve execution efficiency.

4.3.8 Join Query

- [Rule] The depth of a multi-table nested join must be less than 8.

  If the join nesting is too deep, slow SQL statements may be generated. You need to optimize the join nesting at the service layer.
- [Rule] Specify the join condition (ON) of each table to avoid Cartesian product.

  In MySQL, JOIN is equivalent to CROSS JOIN and INNER JOIN. However, in the SQL standards, JOIN is equivalent to INNER JOIN only. Therefore, the ON condition must be used together.
- [Rule] Specify the JOIN mode based on the SQL standards during join. Do not use the JOIN keyword directly. Instead, use CROSS JOIN, INNER JOIN, LEFT JOIN, or RIGHT JOIN.
- [Rule] When performing a multi-table join query, assign aliases to the tables to keep the statement logically clear and easy to maintain.
- [Recommendation] Different columns have different comparison overheads. Use a column type with high efficiency for joined columns.
  - The comparison efficiency of the numeric type is much higher than that of the string type.
  - The comparison efficiency of integers is much higher than that of numeric and floating-point types.
- [Recommendation] The joined columns must be of the same data type to avoid the impact of implicit type conversion on the execution efficiency.
- [Recommendation] Do not use nested subqueries. Use table join if possible because subqueries will generate temporary tables, which greatly affects SQL performance.
- [Recommendation] If a large number of NULL values exist in the joined columns, you are advised to add the IS NOT NULL condition to the WHERE condition to improve the execution efficiency.
- For multi-table join examples, refer to "SQL Reference > SQL Syntax > S > SELECT > Join Queries" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.3.9 Subquery

- [Rule] Do not repeat subquery statements within an SQL query.
- [Recommendation] Do not use subqueries in the SELECT target columns. Otherwise, the plan cannot be pushed down, affecting the execution performance.
- [Recommendation] Do not use scalar subqueries.

  A scalar subquery is a subquery whose result is one value and whose condition expression uses an equal operator. The following example is a statement that does not meet the specifications:

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

  You are advised to split the preceding statement into two SQL statements and execute the subquery first.
- [Recommendation] The nesting depth of a subquery cannot exceed 2.

  Subqueries cause temporary table overhead. Therefore, complex queries must be optimized based on service logic.

4.3.10 Transaction

- [Specification] In non-multi-primary mode, the GTM-free mode does not allow cross-node transactions.

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

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

  ```
  SET enable_twophase_commit = on;
  ```

  Run the following command before the transaction ends:

  ```
  RESET enable_twophase_commit;
  ```

  Or:

  ```
  SET enable_twophase_commit = off;
  ```
- [Specification] Large object operations do not support transactions.

  NOTE

  In non-multi-primary mode, large object operations include creating and dropping databases, as well as running ANALYZE and VACUUM.
- [Rule] When accessing the database through the JDBC client, disable the **autocommit** parameter and explicitly execute COMMIT to finalize transactions.
  - If the **autocommit** parameter is enabled, some parameters (such as **fetchsize**) will become invalid.
  - The service logic must be specified to reduce the dependency on the database.
- [Rule] Do not combine multiple SQL statements into one statement when accessing the database through JDBC.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

Example:

– The following statement does not meet specifications:

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

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

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

4.3.11 SQL Compilation

DDL

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When using the CREATE TABLE AS SELECT ... statement to create a table, employ forcible conversion rather than implicit conversion if data type conversion is involved, as the latter may lead to unexpected outcomes.

For details about DDL syntax, see "SQL Reference > SQL Syntax > DDL Syntax Overview" in *Reference*.

Data Loading and Unloading

- Explicitly set the inserted column list in the INSERT statement. Example:
  ```
  INSERT INTO task(name,id,comment) VALUES ('task1','100','100th task');
  ```
- After data is imported to the database in batches or the data increment reaches the threshold, you are advised to analyze tables to prevent the execution plan from being degraded due to inaccurate statistics.
- To clear all data in a table, you are advised to use TRUNCATE TABLE instead of DELETE TABLE. DELETE TABLE is not efficient and cannot release disk space occupied by the deleted data.

Type Conversion

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

For details about data types, see "SQL Reference > Data Types" in *Reference*.

Query Operation

- Do not return a large number of result sets to a client except the ETL program. If a large result set is returned, consider modifying your service design.
- Perform DDL and DML operations encapsulated in transactions. For example, operations such as TRUNCATE TABLE, UPDATE TABLE, DELETE TABLE, and DROP TABLE cannot be restored once they are committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- During query compilation, you are advised to list all columns to be queried and avoid using SELECT *. Doing so reduces output columns, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- The cost of joining more than three tables or views, especially full joins, is difficult to be estimated. You are advised to use the WITH TABLE AS statement to create interim tables to improve the readability of SQL statements.
- Avoid using Cartesian products or full joins. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Only IS NULL and IS NOT NULL can be used to determine NULL value comparison results. If any other method is used, NULL is returned. For example, NULL instead of expected Boolean values is returned for NULL<>NULL, NULL=NULL, and NULL<>1.
- Do not use count(col) instead of count(*) to count the total number of records in a table. count(*) counts the NULL value (actual rows) while count(col) does not.
- During the execution of count(col), **NULL** rows are counted as 0. During the execution of sum(col), **NULL** is returned if all records are **NULL**. If not all the records are **NULL**, **NULL** rows are counted as 0.
- When using count() for multiple columns, enclose all the column names in parentheses, for example, count((col1,col2,col3)). When multiple columns are used to count the number of NULL record rows, a row is counted even if all the selected columns are NULL. The result is the same as that when count(*) is executed.
- NULL records are not counted when count(distinct col) is used to calculate the number of non-NULL columns that are not repeated.
- If all statistical columns are NULL when count(distinct (col1,col2,...)) is used to count the number of unique values in multiple columns, Null records are also counted, and the records are considered the same.
- Use the connection operator || to replace the concat function for string connection, because the output of the concat function depends on the data type of the strings to be connected. When the execution plan is generated, the value cannot be calculated in advance. As a result, the query performance deteriorates severely.
- Since the execution plans generated by the now function cannot be pushed down, using it to retrieve the current time will significantly degrade query performance. Therefore, use the following **Table 4-10** to replace the now function.

Table 4-10 Time-related macros

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Macro Name	Description	Example
CURRENT_TIMESTAMP(n)	Obtains the current date and time, including the year, month, day, hour, minute, second, and time zone. **NOTE** *n* indicates the number of digits after the decimal point in the time string.	gaussdb=# SELECT CURRENT_TIMESTAMP(6); timestamptz ---------- 2018-02-02 00:39:55.231689+08 (1 row)

- Do not use scalar subquery statements. A scalar subquery is a subquery in the output list of the SELECT statement. In the following example, "SELECT COUNT(*) FROM films f WHERE f.did = s.id" is a scalar subquery statement:
  ```
  SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
  ```
  Scalar subqueries often result in query performance deterioration. During application development, scalar subqueries need to be converted into equivalent table joins based on the service logic.
- In WHERE clauses, the filter conditions should be collated. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.
- Filter conditions in WHERE clauses should comply with unilateral rules, that is, to place the column name on one side of a comparison operator. In this way, the optimizer automatically performs pruning optimization in some scenarios. The format is *col op expression*, where *col* indicates a table column, *op* indicates a comparison operator, such as = and >, and *expression* indicates an expression that does not contain a column name. 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;
  ```
- Do not perform unnecessary collation operations. Collation requires a large amount of memory and CPU. If service logic permits, ORDER BY and LIMIT can be combined to reduce resource overhead. By default, GaussDB performs collation by ASC & NULL LAST.
- When the ORDER BY clause is used for collation, specify collation modes (**ASC** or **DESC**), and use NULL FIRST or NULL LAST for NULL record sorting.
- Do not rely on only the LIMIT clause to return the result set displayed in a specific sequence. Combine ORDER BY and LIMIT clauses if some specific result sets are returned, and use OFFSET to skip specific results if necessary.
- If the service logic is accurate, you are advised to use UNION ALL instead of UNION.
- If a filter condition contains only an OR expression, convert the OR expression to UNION ALL to improve performance. SQL statements that use OR expressions cannot be optimized, resulting in slow execution. For example, the conversion of the following statements:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

Convert the statement to the following:

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

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

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

Rewrite the statement as follows:

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

NOTE

- If the value of the **T1.C1** column is not **NOT NULL**, the preceding rewriting cannot be performed.
- If the **T1.C1** column is the output of a subquery, check whether the output is **NOT NULL** based on the service logic.

- Use cursors instead of the LIMIT OFFSET syntax to perform pagination queries to avoid resource overheads caused by multiple executions. Cursors must be used within transactions. Be sure to close the cursors and commit the transactions once execution completes.

4.4 Client Programming Specifications

4.4.1 JDBC

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

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

These parameters could lead to inconsistent behavior between the JDBC and gsql clients, causing differences in aspects like date display mode, floating-point precision, and time zone.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

In M-compatible databases, the default value of **extra_float_digits** is **0**.

- [Specification] A database must be specified for each JDBC instance. Once an instance is created, it is not possible to alter its database.
- [Specification] The length of a single SQL statement cannot exceed 2 GB. Considering the communications cost, it is recommended that the length of a single SQL statement be less than or equal to 5 KB.
- [Specification] Currently, only the DEFAULT value of the columns in CREATE/ALTER TABLE can be set through parameters. Other DDL statements do not support parameter settings using the Prepare Execute method.
- [Rule] Each PreparedStatement of JDBC can contain a maximum of 32,767 parameters.
- [Specification] The connection parameter **fetchsize** must be used when **autocommit** is disabled. Otherwise, the **fetchsize** configuration is invalid.
- [Rule] Use the default GUC parameters and avoid modifying them by sending SET requests through JDBC. For details, see **GUC Parameter Programming Specifications**.
- [Rule] In non-multi-primary mode, the database uses the GUC parameter **jdbc.version_num** for version control. Applications must not modify this parameter. The **jdbc.version_num** parameter is supported from version 503.1 onwards. Refer to **Table 4-11** for detailed specifications.

Table 4-11 Specifications of the jdbc.version_num parameter

Version	jdbc.version_num	Range
505.2	502*xx*	50200–50299
505.1	501*xx*	50100–50199
505.0	500*xx*	50000–50099
503.1	301*xx*	30100–30199

- [Rule] You must use the Prepare Execute method to execute query statements for higher efficiency.
- [Rule] Execute SQL statements one by one in the same transaction. Do not combine multiple SQL statements and send them as one statement. For details, see **[Rule] Do not combine multiple SQL statements into one statement when accessing the database through JDBC**.
- [Rule] If a temporary table is created in a connection, delete the table before releasing the connection to the connection pool to avoid service errors.
- [Recommendation] In precision-sensitive scenarios, the numeric type is recommended.
- [Recommendation] When connecting to the database through JDBC, ensure that the following three time zones are the same:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Time zone of the host where the JDBC client is located
  - Time zone of the host where the GaussDB cluster is located
  - Time zone during GaussDB cluster configuration

    NOTE

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

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

- **loginTimeout**: integer type. This parameter specifies how long to wait for establishment of a database connection. The timeout is measured in seconds, with a default value of **0** indicating that it is disabled and no timeout will occur.
- **connectTimeout**: integer type. This parameter specifies the timeout value for server connection operations. If connecting to the server takes longer than this value, the connection is broken. The timeout is measured in seconds, with a default value of **0** indicating that it is disabled and no timeout will occur.
- **socketTimeout**: integer type. This parameter specifies the timeout value for socket read operations. If reading from the server takes longer than this value, the connection is closed. The timeout is measured in seconds, with a default value of **0** indicating that it is disabled and no timeout will occur.
- **cancelSignalTimeout**: integer type. The cancel message can itself get stuck. This property controls **connectTimeout** and **socketTimeout** used for cancel commands. The timeout is measured in seconds, with a default value of **10**.
- **tcpKeepAlive**: Boolean type. This parameter is used to enable or disable TCP keepalive probe. The default value is **false**.

4.4.2 Connection Control Specifications

[Rule] You are advised to maintain a static connection pool or control the connection frequency to prevent frequent creation of new connections. If the number of connections per second on a single CN or DN exceeds the recommended limits (below), thread pool usage and CPU usage may surge when the connection thread pool is disabled, potentially impacting service performance. When the connection thread pool is enabled, client connections may experience increased delays or even timeouts.

- An 8-core CPU + 64 GB system can establish a maximum of 90 connections per second with no CPU load, 50 connections at 30% CPU load, 30 connections at 50% CPU load, and 12 connections at 80% CPU load.
- A 16-core CPU + 128 GB system can establish a maximum of 160 connections per second with no CPU load, 90 connections at 30% CPU load, 50 connections at 50% CPU load, and 12 connections at 80% CPU load.
- A 32-core CPU + 256 GB system can establish a maximum of 180 connections per second with no CPU load, 150 connections at 30% CPU load, 50 connections at 50% CPU load, and 12 connections at 80% CPU load.
- A 64-core CPU + 512 GB system can establish a maximum of 180 connections per second with no CPU load, 170 connections at 30% CPU load, 60 connections at 50% CPU load, and 20 connections at 80% CPU load.

4.4.3 C/C++ Application Compilation Specifications

During C/C++ application development, differences in compiler types, language standards, and compilation parameters may introduce undefined behavior, security risks, and platform compatibility issues, potentially compromising system stability and product delivery quality. This specification establishes unified guidelines to streamline compiler selection, language standards, compilation parameters, build modes, and related management requirements for C/C++ applications. It serves as a foundational reference for project development, integration, testing, and delivery.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

To locate memory issues (if any) in C/C++ applications, you can enable AddressSanitizer (ASan) and parse ASan logs. For details, see **ASan** and **Parsing ASan Logs**.

ASan

AddressSanitizer (ASan) is a memory error detector for C/C++ applications. The following describes how to detect memory issues using ASan logs.

Table 4-12 outlines the memory errors detectable by ASan.

Table 4-12 Keywords for Asan-detectable errors

Keywords	Description
heap-buffer-overflow	Heap overflow
heap-use-after-free	Use after deallocation
double-free	Repeated deallocation
invalid-free	Illegal deallocation
stack-buffer-overflow	Stack overflow
stack-use-after-return	Stack use after return (UAR) (stack memory remains occupied even after the function has returned results)
global-buffer-overflow	Global region overflow
alloc-dealloc-mismatch	Allocation-deallocation mismatch
new-delete-type-mismatch	New/Delete error
use-after-poison	Access to red zones
memory-leak	Memory leak

NOTE

This tool supports the x86 and Arm CPU architectures as well as the Linux OS.

Enabling ASan and Generating ASan Logs

The following describes how to enable ASan and generate ASan logs:

Step 1 To use ASan, add **-fsanitize=address** to the compilation command when compiling a C/C++ application.

Below is an example of a basic compilation command:

```
gcc test.c -o test -fsanitize=address
```

Step 2 Check whether the ASan environment variable is set.

```
echo $ASAN_OPTIONS
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If the output is empty, perform **Step 3**. Otherwise, check whether the ASan environment variable includes the target parameters and whether the parameter values are correct.

Step 3 Set the Asan environment variable.

Run the following command to set the Asan environment variable:

```
export ASAN_OPTIONS="unmap_shadow_on_exit=true:abort_on_error=true:log_path=[log_path]/[log_file_prefix]:alloc_dealloc_mismatch=1:fast_unwind_on_fatal=true:detect_stack_use_after_return=1:log_exe_name=true"
```

NOTE

Below is the reference for setting **ASAN_OPTIONS**:

- **unmap_shadow_on_exit**: explicitly unmaps shadow memory (huge pages) on exit. The default value is **false**.
- **abort_on_error**: calls the abort() function rather than _exit() after printing an error report. The default value is **false**.
- **log_path**: writes logs to **log_path.pid**. With two special values (**stdout** and **stderr**), it defaults to **stderr**.

  You are advised to set **log_path** in the environment variable to specify the ASan log output directory. If not set, the specific error stack cannot be obtained. *[log_path]* indicates the log file directory, and *[log_file_prefix]* indicates the log file prefix. Both *[log_path]* and *[log_file_prefix]* can be user-defined.
- **alloc_dealloc_mismatch**: detects memory allocation-deallocation mismatch errors. The default value is **0**.
- **fast_unwind_on_fatal**: uses the fast frame pointer to handle fatal errors. The default value is **false**.
- **detect_stack_use_after_return**: detects stack UAR errors. The default value is **0**.
- **log_exe_name**: mentions the executable file name when reporting an error and adds the file name (for example, **log_path.exe_name.pid**) to logs.

Step 4 Execute the compiled file.

```
./test
```

Step 5 View ASan logs.

Upon completion of execution, ASan logs will be generated in the directory specified by *[log_path]* (which is part of the **log_path** parameter set in the ASan environment variable *ASAN_OPTIONS* in **Step 2**). The log files follow this naming format: *[log_file_prefix].[exe_name].[pid]*.

For example, if **log_path** is set to **/var/log**, **log_file_prefix** is **runlog**, **exe_name** is **test**, and the PID for running **test** is **1234**, the log output directory will be:

```
/var/log/runlog.test.1234
```

----End

Parsing ASan Logs

The following uses an example ASan log to analyze the log structure:

```
=================================================================
==602791==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x603000026904 at pc 0x000000400775 bp 0x7ffeba71f610 sp 0x7ffeba71f608
WRITE of size 4 at 0x603000026904 thread T0
   #0 0x400774 in main (/home/test/workspace/mem+0x400774)
   #1 0x7fb441858964 in __libc_start_main (/lib64/libc.so.6+0x22964)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    #2 0x400658  (/home/test/workspace/mem+0x400658)

0x603000026904 is located 0 bytes to the right of 20-byte region [0x6030000268f0,0x603000026904)
allocated by thread T0 here:
    #0 0x7fb4428d84d0 in __interceptor_malloc ../../.././libsanitizer/asan/asan_malloc_linux.cc:62
    #1 0x400731 in main (/home/test/workspace/mem+0x400731)
    #2 0x7fb441858964 in __libc_start_main (/lib64/libc.so.6+0x22964)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/test/workspace/mem+0x400774) in main
Shadow bytes around the buggy address:
  0x0c067fffccd0: 00 00 00 00 fa fa 00 00 00 00 fa fa 00 00 00 fa
  0x0c067fffcce0: fa fa 00 00 00 00 fa fa 00 00 00 00 fa fa fd fd
  0x0c067fffccf0: fd fd fa fa fd fd fd fd fa fa 00 00 00 fa fa fa
  0x0c067fffcd00: fd fd fd fd fa fa fd fd fd fd fa fa fd fd fd fd
  0x0c067fffcd10: fa fa fd fd fd fd fa fa 00 00 00 fa fa fa 00 00
=>0x0c067fffcd20:[04]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fffcd30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fffcd40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fffcd50: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fffcd60: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fffcd70: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==602791==ABORTING
```

Key sections of the log are parsed as follows, from top to bottom:

Header:

```
==602791==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x603000026904 at pc
0x000000400775 bp 0x7ffeba71f610 sp 0x7ffeba71f608
```

Field explanations:

- ==602791==

  Process ID (PID). ASan uses such IDs to identify the source processes (particularly useful in multi-process scenarios).

- ERROR

  Log level or event type (typically **ERROR**).

- AddressSanitizer

  Name of the sanitizer that generates the report.

- heap-buffer-overflow

  Error type keyword. For details, see **Table 4-12**.

- on address 0x...

  Address of the problematic memory (accessed target address).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- at pc 0x...

  Program counter (PC): instruction address at the time of access (can be symbolized to a function/line number).

- bp 0x... sp 0x...

  Register snapshot: base pointer or stack pointer (used for stack frame locating and debugging).

- WRITE of size 4 at 0x603000026904 thread T0

  Illegal access when the main thread (**T0**) writes 4-byte data to the heap memory address.

The digest of the access action is as follows:

```
WRITE of size 4 at 0x603000026904 thread T0
```

Field explanations:

- WRITE/READ

  Access type (write or read).

- of size 4

  Size of the access (in bytes).

- at 0x...

  Accessed address (same as the first line).

- thread T0

  Thread ID. **T0** typically refers to the main thread, while multi-threaded programs may show T1/T2…

The call stack triggered by an error is as follows:

```
#0 0x400774 in main (/home/test/workspace/mem+0x400774)
#1 0x7fb441858964 in __libc_start_main (/lib64/libc.so.6+0x22964)
#2 0x400658  (/home/test/workspace/mem+0x400658)
```

The address and object boundary are described as follows:

```
0x603000026904 is located 0 bytes to the right of 20-byte region [0x6030000268f0,0x603000026904)
```

- is located ...

  Relative position of the target address to a known memory block (for example, 0 bytes outside the right boundary, *X* bytes outside the left boundary, or at an offset of *X* bytes inside the block).

- xx-byte region

  Size of the related memory block.

- [start, end)

  Start and end addresses of the memory block (typically represented as a left-closed, right-open interval).

Allocation/Deallocation source:

```
allocated by thread T0 here:
   #0 0x... in __interceptor_malloc ...:62
   #1 0x... in main (...)
```

- allocated by thread T0 here

  Thread where the memory was allocated and the specific allocation call, including the stack trace at allocation time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- #0 ... __interceptor_malloc

  Interceptor function. ASan replaces or wraps malloc/free/new/delete with its own versions to record metadata and detect overflows.

  asan_malloc_linux.cc:62 is the source code location of the ASan runtime library (showing this is an ASan implementation of the interceptor).

  In the event of a use-after-free error, the "freed by thread ... here:" stack is also provided.

SUMMARY line (brief summary):

```
SUMMARY: AddressSanitizer: heap-buffer-overflow (/path/to/bin+0x400774) in main
```

It typically includes:

- Sanitizer name.
- Error type.
- A representative location (module + offset, and symbolized function name).

This brief summary helps quickly locate the primary error point.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5 Application Development Guide

5.1 GaussDB Application Development Guide

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

NOTE

When applications use driver-executed SQL statements, it is crucial to ensure that the strings does not contain character 0. Otherwise, execution will fail.

JDBC

Java database connectivity (JDBC) is a set of Java APIs designed to execute SQL statements, integrating unified access APIs for various relational databases. It allows Java applications to use SQL statements to perform database operations, such as querying, inserting, updating, and deleting data.

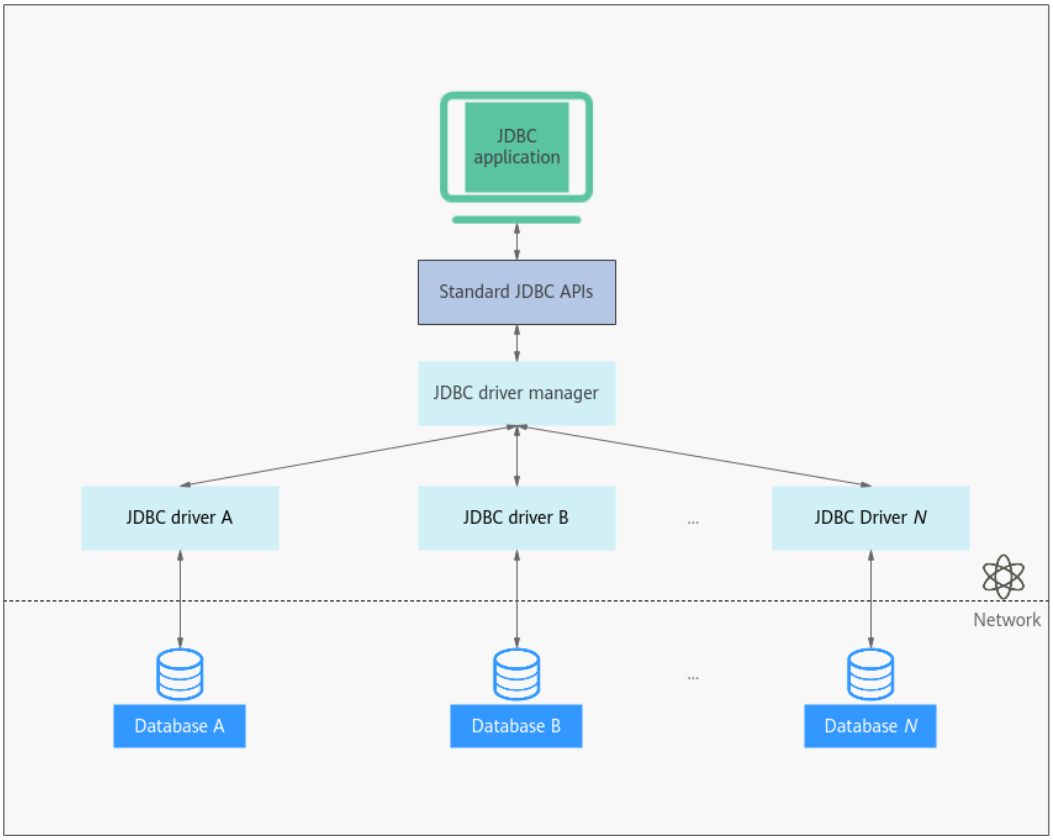
The JDBC architecture primarily comprises the following components:

- JDBC APIs: enable Java applications to interact with databases through standard APIs (such as Connection, Statement, and ResultSet).
- Driver manager: loads and manages JDBC drivers, and selects an appropriate driver based on the connection URL provided in the request.
- Drivers: implement JDBC APIs, translate Java method calls into database-specific protocols, send SQL statements to databases for execution, and return results to applications.

A typical workflow involves a Java application initiating a database operation request (such as a query or update) and communicating with the JDBC driver manager through JDBC APIs. The JDBC driver manager then loads an appropriate JDBC driver. The JDBC driver communicates with a specific database and sends the SQL statement to the database. The database executes the SQL statement and returns a result set. **Figure 5-1** illustrates the JDBC system architecture.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-1 JDBC system architecture



Some key features and usage of JDBC are as follows:

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

In conclusion, JDBC provides a flexible and powerful bridge that enables Java applications to interact with various databases and to implement data persistence and management. GaussDB supports JDBC 4.2. You can compile program code

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

using JDK 1.8, JDK 17, and JDK 21 (version-specific driver packages are required). However, JDBC-ODBC bridging is not supported.

For details about JDBC-based development, see **Development Based on JDBC**.

ODBC

Open database connectivity (ODBC) is a set of standard database access APIs developed by Microsoft in accordance with the Call Level Interface (CLI) specifications from Open Group and ISO/IEC. It uses C/C++ languages to provide unified APIs, enabling applications to achieve cross-platform access to diverse database management systems (DBMSs) by abstracting away the underlying database types or OS differences.

The primary goal of ODBC is to maximize interoperability—the same application source code can access different DBMSs simply by switching drivers. The driver manager processes ODBC function calls from applications and dynamically loads database-specific drivers at runtime. Drivers translate SQL requests into instructions recognizable by specific databases and return execution results to applications. This abstraction mechanism, analogous to printer drivers shielding hardware differences, positions ODBC as a universal layer for database access, enhancing cross-platform capabilities, maintainability, and scalability.

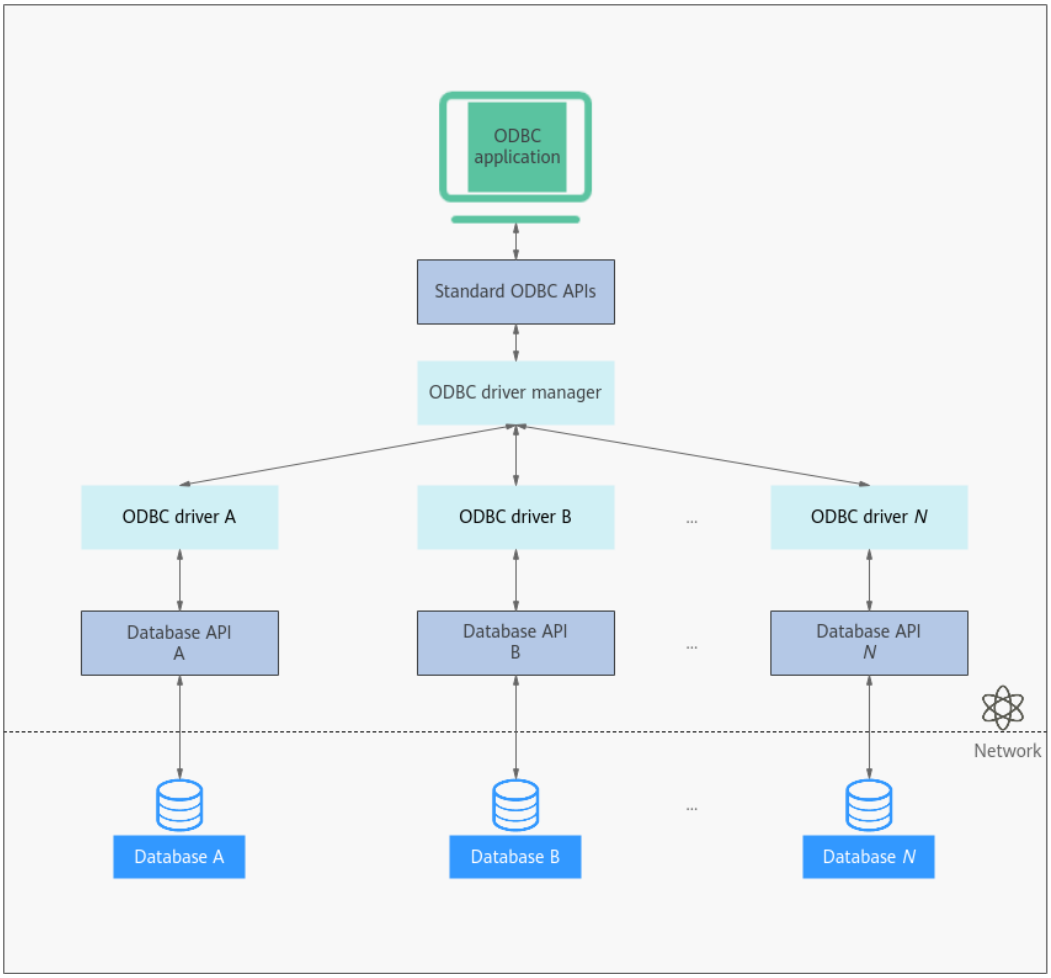
The ODBC architecture primarily comprises the following components:

- Applications: implement service logic and initiate ODBC function calls.
- Driver manager: receives, distributes, and manages drivers.
- Drivers: process ODBC function calls, submit SQL requests to specific data sources, and return results to applications.

Applications connect to the driver manager through standard ODBC APIs, and the driver manager manages the loading and execution of ODBC drivers. **Figure 5-2** illustrates the architecture.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-2 ODBC system architecture



As a database API standard, ODBC is independent of DBMSs, OSs, and programming languages. ODBC APIs comply with the CLI specifications defined by Open Group and ISO/IEC, with ODBC 3.x fully implementing these specifications. GaussDB supports ODBC 3.5 and provides drivers for both Linux/Unix and Windows environments:

- The unixODBC-2.3.7 driver manager is designed for Unix/Linux environments.
- The Windows OS utilizes its built-in ODBC driver manager, accessible through **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

Table 5-1 lists the supported Linux/Unix environments.

Table 5-1 Supported Linux/Unix environments

OS	Platform
EulerOS V2.0SP5	x86_64
EulerOS V2.0SP9	ARM64
EulerOS V2.0SP10	x86_64

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

OS	Platform
EulerOS V2.0SP10	ARM64
Kylin V10	x86_64
Kylin V10	ARM64
UnionTech V20	x86_64
UnionTech V20	ARM64
Huawei Cloud EulerOS 2.0	x86_64
Huawei Cloud EulerOS 2.0	ARM64
SUSE Linux Enterprise Server 12 SP5 **NOTE** SUSE Linux Enterprise Server 12 SP5 does not include the dynamic library libltdl by default. Install libltdl before using ODBC.	x86_64
openEuler 22.03 LTS SP4	ARM64
BCLinux-R8-U2	x86_64
BCLINUX21.10	x86_64
REDHAT 8.10	x86_64
REDHAT 8.10	ARM64
NFSChina Server 4.0	x86_64
NFSChina Server 4.0	ARM64
Ubuntu 22.04 LTS	x86_64
Ubuntu 22.04 LTS	ARM64
Suse15.7	x86_64
Suse15.7	ARM64
Huawei Cloud EulerOS 3.0	x86_64
Huawei Cloud EulerOS 3.0	ARM64
Kylin V11	ARM64

Table 5-2 lists the supported Windows environments.

Table 5-2 Supported Windows environments

OS	Platform
Windows 7	x86_32

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

OS	Platform
Windows 7	x86_64
Windows Server 2008	x86_32
Windows Server 2008	x86_64

NOTICE

When using ODBC to access GaussDB, be aware of the following limitations present in the current ODBC driver of GaussDB, in addition to standard ODBC behaviors (these limitations may impact the use of certain SQL functions or data types):

- ODBC does not support user-defined types, nor does it support using parameters of user-defined types within stored procedures.
- When **proc_outparam_override** is enabled in a database (that is, the GUC parameter **behavior_compat_options** includes **proc_outparam_override**), ODBC cannot call stored procedures that involve OUT parameters.
- ODBC does not support read-on-standby in the standby cluster.
- When working with hash bucket tables, slice bucket tables, or replication tables, or performing SQL_UPDATE operations, ODBC does not support updatable cursors.
- The ODBC driver is based on an open-source version and may encounter compatibility issues with data types such as TINYINT, SMALLDATETIME, and NVARCHAR2.

For details about ODBC-based development, see **Development Based on ODBC**.

libpq

libpq is a C-language client API library provided by GaussDB, serving as the primary development interface for C/C++ applications to access GaussDB.

libpq encapsulates the complete communication protocol between clients and the database server, offering developers a set of feature-rich and high-performance APIs. Through libpq, applications can establish and manage database connections, execute SQL statements, process result sets, control transactions, and perform various advanced database operations. Its lightweight design makes it particularly suitable for service scenarios that require high performance, concurrency, and resource efficiency. Additionally, libpq acts as the underlying engine for several other GaussDB APIs, such as ODBC, psycopg, and ECPG.

The system architecture of libpq consists of two main components:

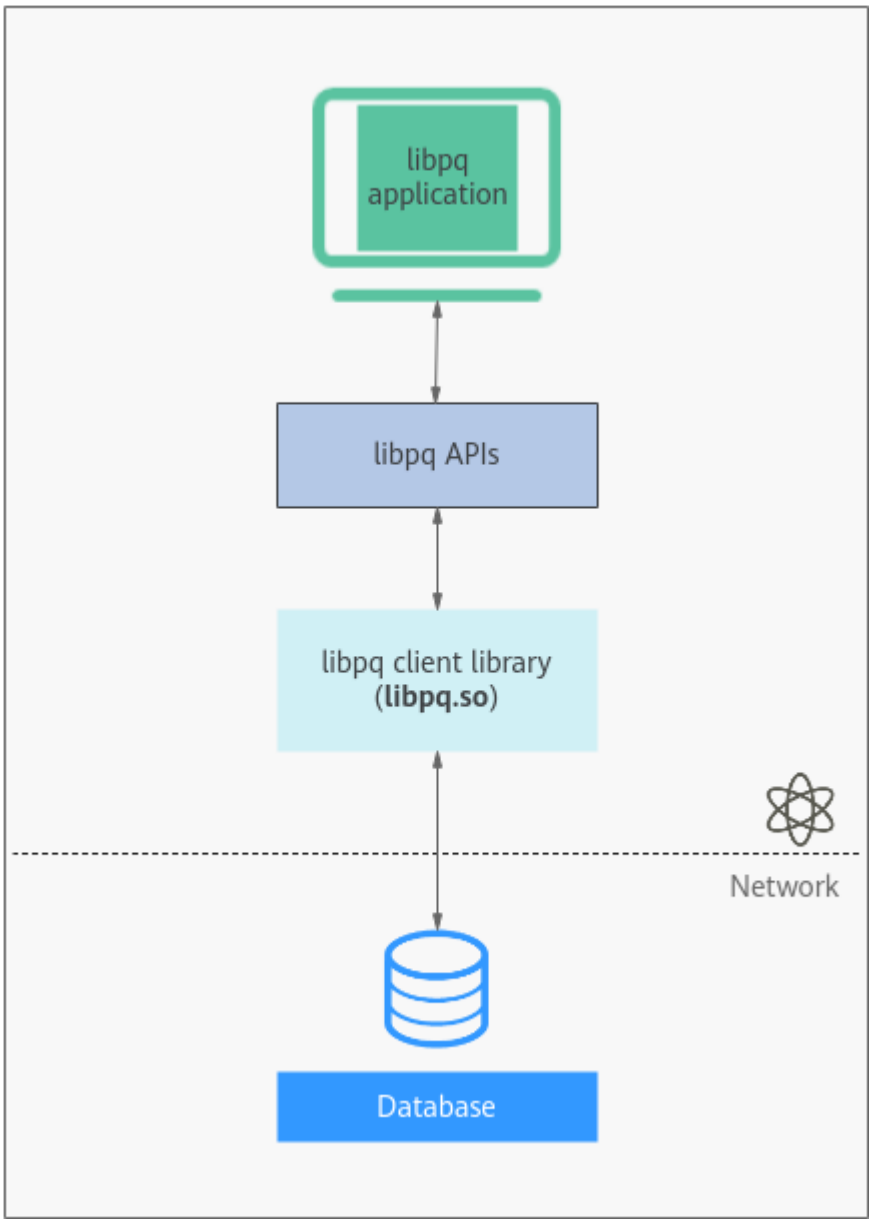
Application: a C/C++ program written by developers, performing all database operations by calling the APIs provided by libpq.

libpq client library: the core API library (**libpq.so**), responsible for implementing APIs, encapsulating protocols, maintaining connection status, encoding/decoding data, and handling errors.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The system architecture of libpq is illustrated in the following figure.

Figure 5-3 System architecture of libpq



The OS platforms supported by libpq align with those supported by GaussDB and are updated in sync with GaussDB versions.

NOTICE

libpq itself is not thread-safe. In multi-threaded applications, it is advisable to use a dedicated connection per thread or employ an external connection pool for management.

For details about libpq-based development, see **Development Based on libpq**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Psycopg

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

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

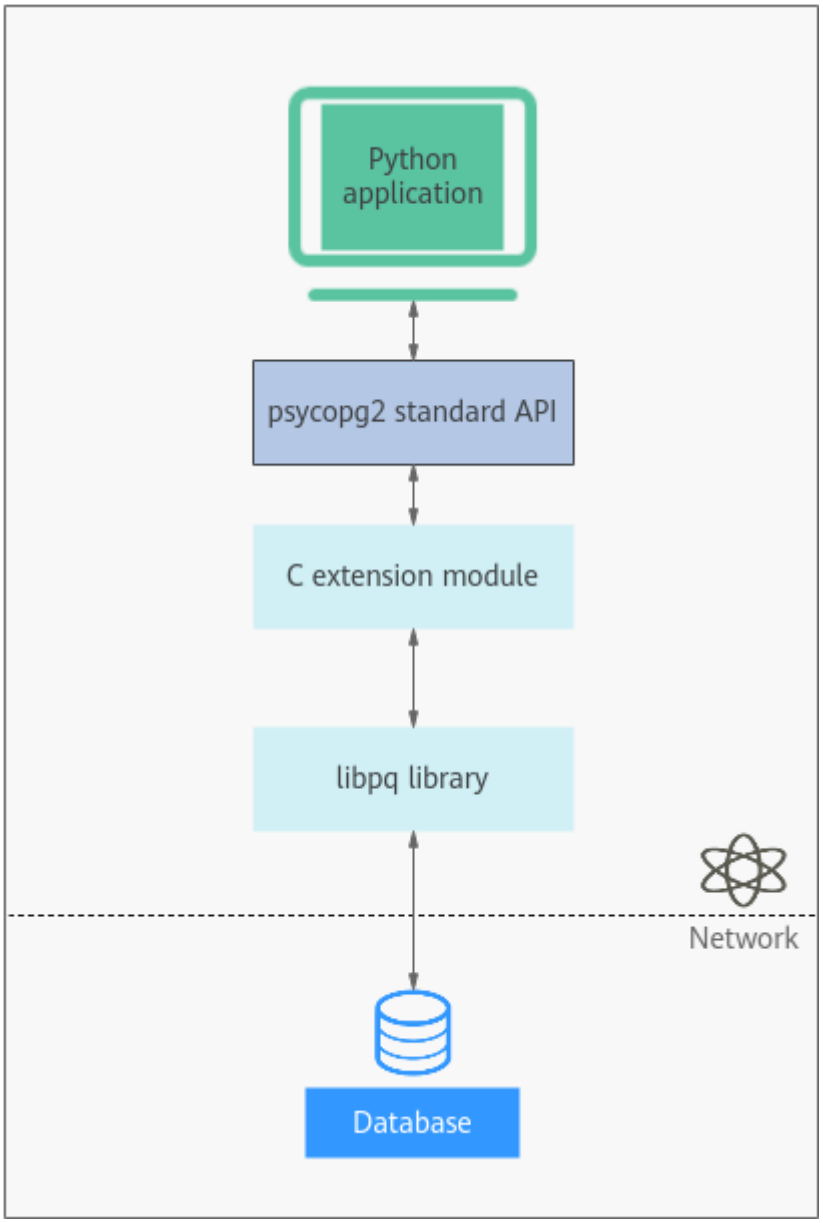
The system architecture of psycopg consists of four main components:

- Application: implements the service logic. An application initiates database operation requests and receives returned data results by calling standard Python objects and methods.
- psycopg2 standard API: implements the Python database API specification (DB-API 2.0). An API standardizes requests (such as requests for executing SQL statements and processing query results) from Python applications and passes them to the C extension module.
- C extension module: quickly converts Python data types and objects into underlying C data structures and directly calls libpq functions.
- libpq library: serves as an underlying C dynamic link library that establishes TCP connections with the database server and exchanges network protocol packets.

Figure 5-4 shows the system architecture of psycopg.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-4 System architecture of psycopg



Currently, only the Linux/Unix environment is supported. For details, see **Table 5-3**.

Table 5-3 Platforms supported by psycopg

OS	Platform	Python Version
EulerOS V2.0SP5	• x86_64	3.11.4
EulerOS V2.0SP9	• ARM64	3.11.4
EulerOS V2.0SP10	• ARM64	3.11.4
EulerOS V2.0SP10	• x86_64	3.11.4
openEuler 22.03 LTS SP4	• ARM64	3.11.4

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

OS	Platform	Python Version
Huawei Cloud EulerOS 2.0	• ARM64 • x86_64	3.11.4
Kylin V10	• ARM64 • x86_64	3.11.4
UnionTech20	• ARM64 • x86_64	3.11.4
SUSE Linux Enterprise Server 12 SP5	• x86_64	3.11.4
BCLinux-R8-U2	• x86_64	3.11.4
REDHAT 8.10	• ARM64 • x86_64	3.11.4
BCLINUX 21.10	• ARM64 • x86_64	3.11.4
NFSChina Server 4.0	• ARM64 • x86_64	3.11.4
Ubuntu 22.04 LTS	• ARM64 • x86_64	3.11.4
Suse 15.7	• ARM64 • x86_64	3.11.4
Huawei Cloud EulerOS 3.0	• ARM64 • x86_64	3.11.4
Kylin v11	• ARM64	3.11.4

NOTICE

During compilation, psycopg2 is linked to GaussDB's OpenSSL, which may be incompatible with the OS-provided OpenSSL. If such incompatibility occurs, for example, if "version 'OPENSSL_1_1_1f' not found" is displayed, set the environment variable *LD_LIBRARY_PATH* to isolate GaussDB's OpenSSL from the OS-provided OpenSSL.

For example, when running the **client.py** application that calls psycopg2, explicitly assign the environment variable to the application as follows:

```
export LD_LIBRARY_PATH=/path/to/psycopg2/lib:$LD_LIBRARY_PATH python client.py
```

In the preceding command, **/path/to/psycopg2/lib** indicates the directory storing the OpenSSL library that GaussDB depends on. Change it to the actual directory.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about psycopg-based development, see **Development Based on Psycopg**.

Go

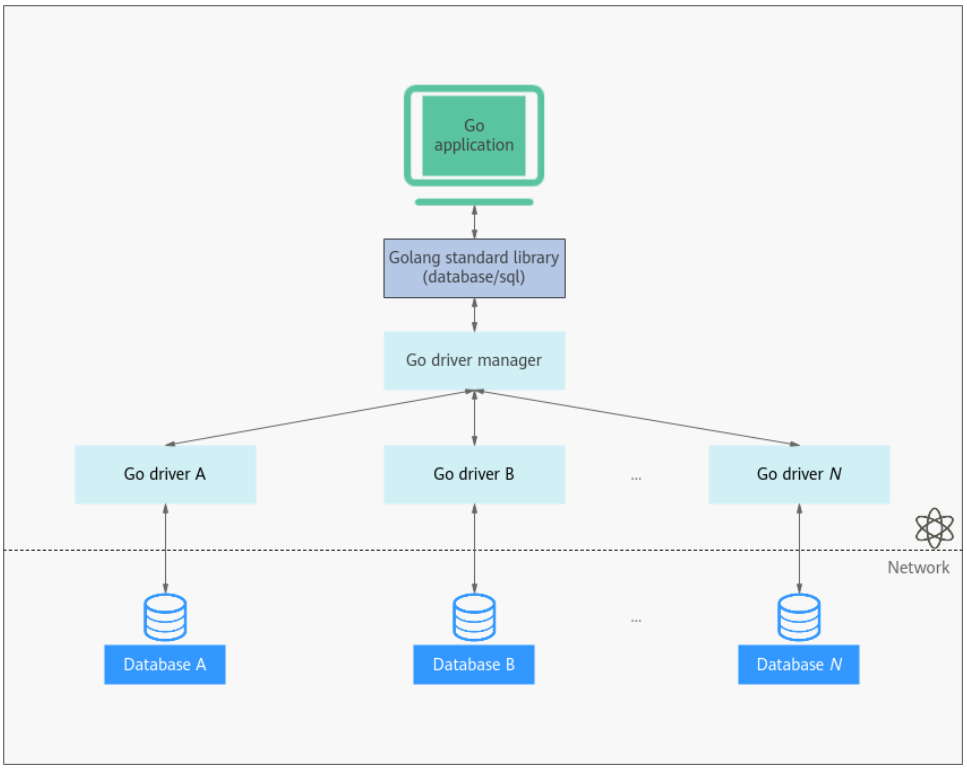
Go language driver (Go driver) used to connect to and operate on GaussDB. It provides some APIs for connecting to and operating on GaussDB. You can use these APIs to perform operations such as query, insert, update, and delete.

The Go architecture primarily comprises the following components:

- Application: implements the service logic by initiating database operation requests (such as SQL queries and transaction processing).
- Golang standard library (database/sql): implements the abstraction layer of the Go library, which defines unified database operation APIs (such as sql.Open and type DB).
- Driver manager: loads and manages Go drivers, and selects an appropriate driver based on the **driverName** parameter of the sql.Open API in database/sql.
- Driver: implements database/sql database operation APIs, translates Go method calls into database-specific protocols, sends SQL statements to databases for execution, and returns results to applications.

Figure 5-5 shows the system architecture of Go.

Figure 5-5 System architecture of Go



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

1. Database connection: A Go driver can connect to a database through db.Open. The connection string can be in URL or DSN format.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

For details about Go-based development, see **Development Based on the Go Driver**.

ECPG

The embedded SQL C preprocessor for GaussDB (ECPG) offers a standard approach for integrating SQL capabilities directly into C programs. ECPG allows developers to embed standard SQL statements in C source code by using the **EXEC SQL** prefix.

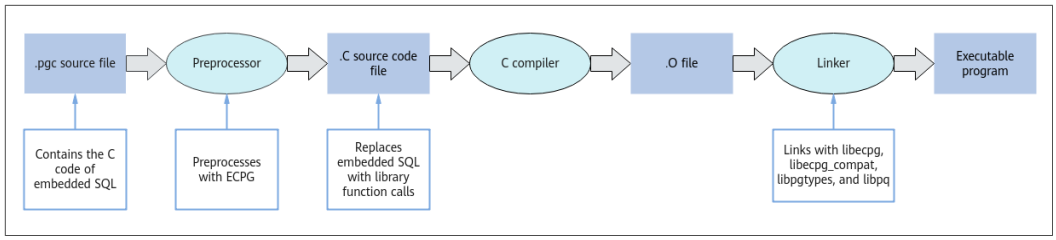
This special code always has the form:

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

These statements syntactically take the place of a C statement. Depending on the particular statement, they can appear at the global level or within a function. The core value of ECPG lies in its use of preprocessing techniques to abstract the complexity of underlying communication and protocol processing, achieving deep syntactic integration between the host language (C) and database language (SQL).

ECPG operates through a typical layered architecture of "preprocess -> compile -> link." Its core workflow is illustrated in **Figure 5-6**.

Figure 5-6 ECPG architecture



As shown above, an embedded SQL program consists of code written in an ordinary programming language, in this case C, mixed with SQL commands. To build the program, the source code (*.pgc) is first passed through the embedded SQL preprocessor, which converts it to an ordinary C program (*.c), and afterwards it can be processed by a compiler, converting the embedded logic into an .O file. Converted ECPG applications call functions in the libpq library through the embedded SQL library (ecpglib), and communicate with the GaussDB server using the normal frontend-backend protocol.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The core advantages of ECPG include:

- **Development efficiency:** Through deep integration of SQL and C, ECPG eliminates cumbersome API call logic, leading to less code and more intuitive maintenance.
- **Data security:** ECPG automatically processes data exchange logic between C programs and databases, ensuring accurate data type mapping during transmission. This mechanism effectively mitigates risks from manual data conversion and inherently resists SQL injection attacks through parameterized processing.
- **Early warning:** Unlike traditional APIs that report errors only at runtime, ECPG performs static syntax analysis on embedded SQL statements, allowing developers to catch most syntax errors during the build phase. This reduces debugging costs and improves software robustness.

When developing ECPG-based programs, adhere to the following syntax conventions to distinguish SQL logic from C logic:

- Embedded SQL statements follow standard SQL case-sensitivity rules and permit nested C code-style comments (as part of the SQL standard).
- The C portion of the program conforms to standard C conventions and does not support nested comments.

For details about ECPG-based development, see **Development Based on ECPG**.

5.2 Development Specifications

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

- If GUC parameters are set in the connection, you must use the following statement to clear the connection status before returning the connection to the connection pool:
```
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
```
- If a temporary table is used, delete the temporary table before you return the connection to the connection pool.

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

Table 5-4 describes the compatibility of application development drivers.

Table 5-4 Compatibility description

Driver	Compatibility Description
JDBC, Go, ODBC, libpq, Psycopg, and ECPG	The drivers remain compatible with earlier database versions, but unlocking the new capabilities introduced in both the drivers and the databases requires upgrading your databases.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- In principle, you need to set the compatibility parameters after the database is created, rather than switching them when using the database.
- Upgrade the JDBC driver to version 503.1 or later for the non-multi-primary mode if you need the following features:
  1. User-defined nested types, such as record, array, and tableof, in JDBC
  2. Validity check of **sessiontimezone** when the compatibility parameter is set to **s2**
- If the driver of an earlier version is connected to the database of a later version that includes new features, exceptions may occur. You are advised to use the driver and database of the matching version.
- The JDBC driver is not thread-safe in multi-threaded environments. Therefore, avoid using the same connection instance in such environments. The caller must synchronize the calls to the driver.

5.3 Development Based on JDBC

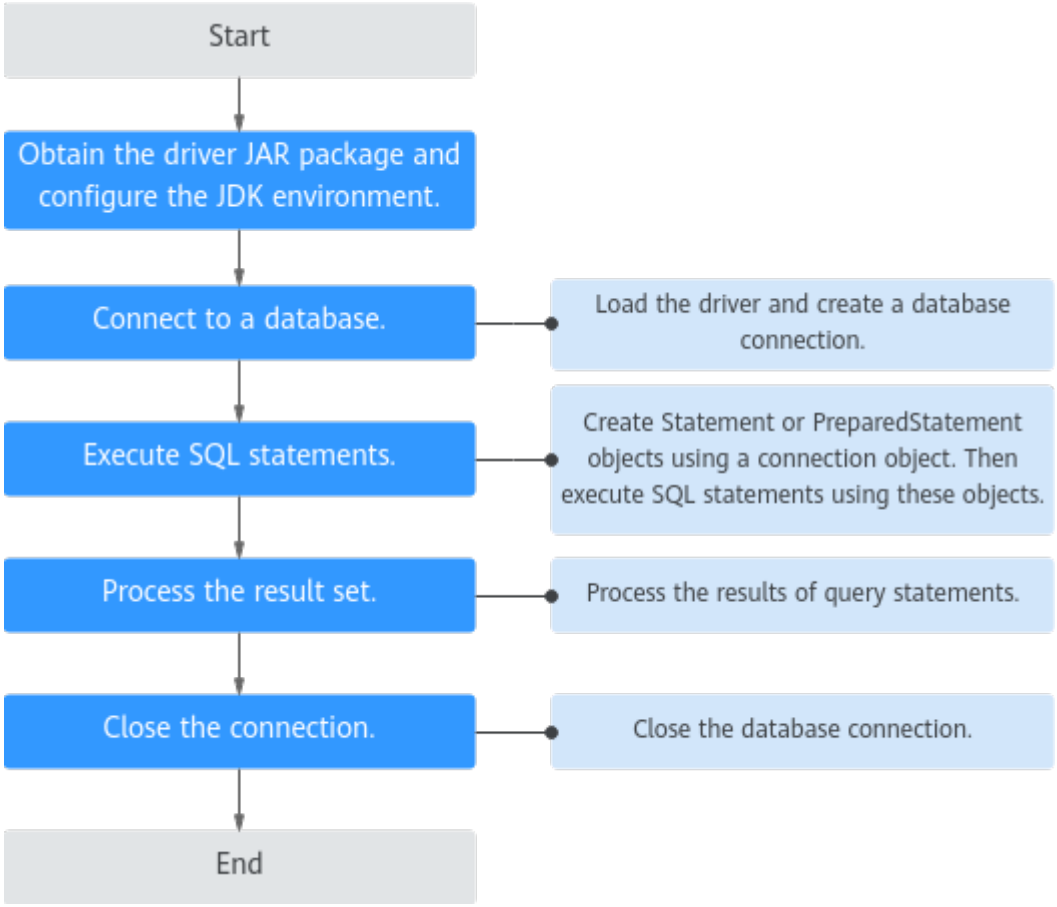
5.3.1 Development Process

The application development process based on JDBC involves these steps: obtain the driver JAR package and configure the JDK environment, connect to a database, execute SQL statements, process the result set, and close the connection.

Figure 5-7 shows the application development process based on JDBC.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-7 Application development process based on JDBC



5.3.2 Development Procedure

5.3.2.1 Obtaining the Driver JAR Package and Configuring the JDK Environment

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

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 5-5**.

Table 5-5 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

- Verifying the software package integrity on a Linux server:
  a. Upload the software package and verification package to the same directory on the VM.
  b. Run the following command to verify the integrity of the software package:
     ```
     F=$(echo DBS-GaussDB-driver*.tar.gz);echo "$(awk '{print $1}' "$F.sha256")  $F" | sha256sum -c -
     ```
     If **OK** is displayed in the command output, the verification is successful. Example:
     ```
     DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz: OK
     ```
- Verifying the software package integrity on a Windows server:
  a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
  b. Run the following command to obtain the hash value of the driver package:

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

     - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
     - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz**.

     Example: **certutil -hashfile** *C:\Users\DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz* **sha256**
  c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-5**.
     - If they are consistent, the verification is successful.
     - If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining the Driver JAR Package

The following describes how to obtain the driver JAR package:

Step 1 Decompress the obtained driver package, which includes various driver JAR packages. Follow the steps below to find the file path for the specific package you need. The following is the standard path hierarchy for a driver JAR package:

DBS-GaussDB-driver_*/*Database deployment mode***/***Driver type***/***OS architecture***/***Release package***/***Driver JAR package*

Step 2 Select a database deployment mode.

The JDBC driver package is independent of the deployment mode. You can use the driver JAR package for any of the deployment modes: Centralized or Distributed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 3 Select a driver type.

Select **jdbc_driver.tar.gz** and decompress it.

Step 4 Select an OS architecture.

In non-multi-primary mode:

1. Non-encrypted functionalities are independent of OS architectures, giving you the freedom to select any OS architecture.
2. Encrypted functionalities are specific to OS architectures. To determine your client OS architecture, run the following command:
   ```
   uname -a
   ```

Step 5 Select a release package.

Run the following command to determine the JDK version of the current environment (this version influences the selection of the appropriate release package):

```
java -version
```

- The JDK 8 release package includes the driver JAR package intended for JDK 1.8.
- The JDK 17 release package includes the driver JAR package intended for JDK 17.
- The JDK 21 release package includes the driver JAR package intended for JDK 21.

Step 6 Select a driver package based on your service requirements.

NOTE

- You are advised to use **gaussdbjdbc.jar** in non-encrypted scenarios and **gscejdbc.jar** in encrypted scenarios. **opengaussjdbc-ce.jar**, **opengaussjdbc.jar**, **gsjdbc4.jar**, and **gsjdbc200.jar** will not be maintained in the future.
- In non-encrypted scenarios, **gaussdbjdbc.jar** is compatible with **opengaussjdbc.jar**, **gsjdbc4.jar**, and **gsjdbc200.jar**. Therefore, when using **gaussdbjdbc.jar** to replace **opengaussjdbc.jar**, **gsjdbc4.jar**, or **gsjdbc200.jar**, you do not need to modify JDBC-related service code.
- In encrypted scenarios, **gscejdbc.jar** is compatible with **opengaussjdbc-ce.jar**. Therefore, when using **gscejdbc.jar** to replace **opengaussjdbc-ce.jar**, you do not need to modify JDBC-related service code.

Decompress the release package to access the driver JAR packages designed for different compatibility modes. All driver JAR packages, except **gscejdbc.jar** and **opengaussjdbc-ce.jar**, offer the same functionalities. The key differences lie in the driver's main class name, connection string prefix, and internal class path. For details, see **Table 5-6**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-6 Driver JAR packages

Driver Package	Connection String Prefix	Driver's Main Class Name	Usage Scenario
gaussdbjdbc.jar	jdbc:gaussdb	com.huawei.gaussdb.jdbc.Driver	The Java code examples in this section use the **gaussdbjdbc.jar** package by default. This driver package is recommended for non-encrypted databases.
opengaussjdbc.jar	jdbc:opengauss	com.huawei.opengauss.jdbc.Driver	This driver package is designed for scenarios where PG databases and GaussDB need to be accessed within the same JVM process.
gsjdbc4.jar	jdbc:postgresql	org.postgresql.Driver	This driver package is designed for scenarios where services are migrated from PG databases. The driver class and loading path remain the same as before the migration, but the supported APIs vary. Any unsupported APIs will need to be adjusted on the service side.
gsjdbc200.jar	jdbc:gaussdb	com.huawei.gauss200.jdbc.Driver	This driver package is designed for scenarios where services are migrated from Gauss200. The driver class and loading path remain the same as before the migration, but the supported APIs vary. Any unsupported APIs will need to be adjusted on the service side. The rest is the same as **gsjdbc4.jar**.
gscejdbc.jar	jdbc:gaussdb	com.huawei.gaussdb.jdbc.Driver	This driver package is recommended for encrypted databases. It includes the encryption/decryption dependency libraries required by encrypted databases. Currently, it applies only to EulerOS and HCE OS.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Driver Package	Connection String Prefix	Driver's Main Class Name	Usage Scenario
opengaussjdbc-ce.jar	jdbc:opengauss	com.huawei.opengauss.jdbc.Driver	This driver package is recommended for encrypted databases with a URL prefix of **jdbc:opengauss**. It includes the encryption/decryption dependency libraries required by encrypted databases. Currently, it applies only to EulerOS and HCE OS.

NOTICE

- GaussDB's driver packages cannot be used for PG databases. While a successful connection may be established in some versions, the behavior of certain APIs is incompatible with PG JDBC, potentially leading to unforeseen errors.
- Similarly, PG's driver packages cannot be used for GaussDB. While a successful connection may be established in some versions, the behavior of certain APIs is incompatible with GaussDB JDBC, potentially leading to unforeseen errors.
- Before using the driver package **gscejdbc.jar**, you need to set the environment variable *LD_LIBRARY_PATH*. For details, see **Using JDBC to Operate on an Encrypted Database**.

Step 7 Add the driver JAR package to the runtime environment.

For example, to develop applications with an IDE in a non-encrypted database, you need to add the **gaussdbjdbc.jar** package (recommended) to the local IDE.

----End

Configuring the JDK Environment

The JDK environment must be configured on the client. JDK supports various platforms such as Windows and Linux. This section uses JDK 1.8 and Windows 11 as an example to describe the specific procedure.

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

```
java -version
```

Check whether JDK 1.8 has been installed. If not, download it from the **official website** and install it.

Step 2 Configure system environment variables.

1. Right-click **This PC** and choose **Properties**.
2. In the **System** > **About** window, click **Advanced system settings**.
3. In the **System Properties** dialog box, click **Environment Variables** on the **Advanced** tab page.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4. In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see **Table 5-7**.

Table 5-7 Variables

Variable	Operation	Variable Value
JAVA_HOME	– If the variable exists, click **Edit**. – If the variable does not exist, click **New**.	Specifies the Java installation directory. Example: **C:\Program Files\Java\jdk1.8.0_131**
Path	Click **Edit**.	– If *JAVA_HOME* is configured, add *%JAVA_HOME%***\bin** before the variable value. – If *JAVA_HOME* is not configured, add the following full Java installation path before the variable value: C:\Program Files\Java\jdk1.8.0_131\bin
CLASSPATH	Click **New**.	*%JAVA_HOME%***\lib;***%JAVA_HOME%***\lib\tools.jar**

----End

5.3.2.2 Connecting to a Database

5.3.2.2.1 Connection Methods

If the driver JAR package has been obtained and the JDK environment has been configured, you still need to load the driver before connecting to a database using JDBC. This section describes the driver loading methods, connection APIs, and connection methods.

Driver Loading Methods

There are two driver loading methods:

- Load the driver at any position in the code before creating a connection.
  ```
  Class.forName("com.huawei.gaussdb.jdbc.Driver");
  ```
- Pass parameters when a JVM is started. **jdbctest** is the name of a test program.
  ```
  java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Since **gsjdbc4.jar** shares the same classpath as PG's JDBC driver, using both **gsjdbc4.jar** and PG's JDBC driver package in the same process may result in class name conflicts.
- In this version, JDBC no longer supports IAM authentication.
- GaussDB's JDBC driver provides the following enhanced features:
  - SHA-256 login.
  - SM3 login. You need to configure the plug-in required for SM3 encryption. For example, JDK 8 requires the following configuration in the **pom.xml** file:

```
<dependency>
<groupId>org.bouncycastle</groupId>
<artifactId>bcprov-jdk18on</artifactId>
<version>1.75</version>
<scope>compile</scope>
</dependency>
```

  - Interconnection with the third-party log framework that implements the SLF4J API.
  - Distributed load balancing at the connection level.
  - DR switchover.

Connection APIs

JDBC provides three APIs for creating database connections, as detailed in **Table 5-8**.

Table 5-8 Database connection APIs

API	Method Signature	Description in Non-Multi-Primary Mode
API 1	DriverManager.getConnection(String url)	You are required to write the database username and password into **url**, which poses security risks and is therefore not recommended.
API 2	DriverManager.getConnection(String url, String user, String password)	The database username and password are passed via separate parameters, as detailed in **Connecting to a Database Using API 2 (Non-Multi-Primary)**.
API 3	DriverManager.getConnection(String url, Properties info)	The database username and password are passed via the **info** parameter, as detailed in **Connecting to a Database Using API 3 (Non-Multi-Primary)**.

For details about the **url**, **info**, **user**, and **password** parameters, see **Table 5-9**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-9 Database connection parameters

Parameter	Description
url	Connection descriptor used to connect to the database through the driver package **gaussdbjdbc.jar**. If **host** is a server name or IPv4 address, acceptable formats are as follows: ● jdbc:gaussdb: ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database ● jdbc:gaussdb://host:port/database ● jdbc:gaussdb://host:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 If **host** is an IPv6 address, acceptable formats are as follows: ● jdbc:gaussdb: ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database or jdbc:gaussdb://[host]/database ● jdbc:gaussdb://[host]:port/database ● jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://[host1]:port1,[host2]:port2/database?param1=value1&param2=value2

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
	NOTE • **database** indicates the name of the database to connect to. The database name is the same as the username by default. • In non-multi-primary mode, **host** refers to the name or IP address of the database server. Both IPv4 and IPv6 addresses are supported. In non-multi-primary mode, database CNs block unauthenticated access from other nodes within the cluster to maintain security. To access a CN from inside the cluster, deploy the JDBC program on the host where the CN is located and set **host** to **127.0.0.1**. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. To prevent any impact on database performance, it is advisable to deploy service systems outside the cluster. By default, connections are established with **localhost**. • In non-multi-primary mode, **port** refers to the port number of the database server. By default, connections are established with port 5432. • If **host** is set to an IPv6 address and a port number is specified in **url**, enclose the IP address in square brackets ([]), as in the *[IP address]:port* format. • **param** indicates the name of a parameter (database connection property). Parameters can be configured in **url**. **url** starts with a question mark (?), uses equal signs (=) to assign values to parameters, and uses ampersands (&) to separate parameters. • **value** indicates the value of the parameter (database connection property). • **connectTimeout** and **socketTimeout** must be set for connections. Otherwise, they will default to **0**, meaning the connections will never time out. In the event of a network failure between the server and the client, the client will be unable to receive ACK packets from the server. The timeout-and-retransmission mechanism will be activated to retransmit the packets until the client successfully receives them. A timeout error will be reported when the packets have been retransmitted 15 times (the default setting), resulting in a high RTO. • When establishing a connection through the standard JDBC API, verify that the **url** format is valid. An invalid **url** can trigger an exception that includes the original **url** string, potentially leaking sensitive information. • You are advised not to include sensitive information such as passwords or credentials in **url**. Instead, configure this information in the connection properties. For details, see **Connecting to a Database Using API 3 (Non-Multi-Primary)**.
info	Database connection properties. For details, see **Connection Parameter Reference**. **NOTICE** If the same parameter is specified in both **url** and **info**, the one in **url** will take precedence.
user	Name of the database user.
password	Password for the database user.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Connection Methods

The database connection method is usually selected based on factors such as security, performance, scalability, and implementation complexity. **Table 5-10** describes the advantages and disadvantages of each connection method.

Table 5-10 Connection methods

Connection Method	Advantages	Disadvantages	Selection Principle
Non-encrypted	• This method is easy to implement and does not require additional encryption/decryption. • It offers high performance because no encryption/decryption is required, reducing data transmission overhead.	• It provides low security, and data is easily intercepted or tampered with during transmission. • It is not suitable for sensitive data transmission and may violate privacy protection laws and regulations.	It applies only to internal networks or scenarios with low security requirements.
SSL	• It provides encrypted transmission. Data is protected during transmission, ensuring high security. • It supports client/server identity authentication, which enhances security.	• It is complex in configuration and management, and certificates need to be issued and updated. • When a large amount of data is encrypted/decrypted, additional performance overhead may be introduced.	It applies to scenarios that require high data transmission security, such as the finance and healthcare industries.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Connection Method	Advantages	Disadvantages	Selection Principle
UDS	• It applies to local communication without passing through the network, featuring a high transmission speed. • No extra network overhead is required, and it is not vulnerable to network attacks.	• It is used only for the communication between processes on the same host but cannot be used for remote connections.	It applies to local communication scenarios, for example, data exchange between local services and applications.

Initial Packet During Connection Establishment

After a socket is established during the connection process between the driver and the database, the first packet sent by the driver to the database, in key-value format, is called the initial packet. This packet includes the username (**user**), database name (**database**), and certain GUC parameters (**client_encoding**, **DateStyle**, **application_name**, **TimeZone**, **extra_float_digits**, **gaussdb.capacity_version**, **replication**, **search_path**, **enable_full_encryption**, **session_type**, and **uppercase_attribute_name**). For detailed descriptions of these GUC parameters, refer to "Database GUC Parameters > GUC Parameters" in *Reference*.

NOTICE

- The **gaussdb.capacity_version** parameter indicates the version number of the compatibility capability set supported by the driver. It is used by the driver and the database to negotiate the compatibility feature set that will be applied during this connection. This is an internal parameter.
- After the database executes the following command, the values of the preceding GUC parameters in the initial packet will be updated to the values specified in the initial packet:

```
reset all;
```

5.3.2.2.2 Connection in Non-encrypted Mode

Connecting to a Database Using API 2 (Non-Multi-Primary)

Prerequisites: The **gaussdbjdbc.jar** package has been added to your runtime environment (such as an IDE).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Use the DriverManager.getConnection(String url, String user, String password) API to create a database connection, as follows:

Step 1 Import java.sql.Connection and java.sql.DriverManager.

java.sql.Connection is a database connection API. It uses the getConnection() method of java.sql.DriverManager to connect applications to a database. You may also import other APIs and classes as needed. For details, see **JDBC API Reference**.

```
import java.sql.Connection;
import java.sql.DriverManager;
```

Step 2 Specify the database **sourceURL** (provide actual values for *$ip*, *$port*, and *database*), username, and password.

Writing the username and password directly in the code creates significant security risks. It is advisable to store them in environment variables instead.

```
String sourceURL = "jdbc:gaussdb://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Load the driver.

Load the database driver **com.huawei.gaussdb.jdbc.Driver**.

```
String driver = "com.huawei.gaussdb.jdbc.Driver";
Class.forName(driver);
```

Step 4 Create a database connection.

Call DriverManager.getConnection(String url, String user, String password) to connect to the database.

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

----End

Connecting to a Database Using API 3 (Non-Multi-Primary)

Prerequisites: The **gaussdbjdbc.jar** package has been added to your runtime environment (such as an IDE).

Use the DriverManager.getConnection(String url, Properties info) API to create a database connection, as follows:

Step 1 Import java.sql.Connection, java.sql.DriverManager, and java.util.Properties.

The setProperty() method of java.util.Properties is used to assign values to Properties objects. You may also import other APIs and classes as needed. For details, see **JDBC API Reference**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the database **sourceURL** (provide actual values for *$ip*, *$port*, and *database*), username, and password.

Writing the username and password directly in the code creates significant security risks. It is advisable to store them in environment variables instead.

```
String sourceURL = "jdbc:gaussdb://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Create a Properties object, and set **userName** and **password** as its property values.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Properties info = new Properties();
info.setProperty("user", userName);
info.setProperty("password", password);
```

Step 4 Load the driver.

Load the database driver **com.huawei.gaussdb.jdbc.Driver**.

```
String driver = "com.huawei.gaussdb.jdbc.Driver";
Class.forName(driver);
```

Step 5 Create a database connection.

Call DriverManager.getConnection(String url, Properties info) to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL, info);
```

----End

5.3.2.2.3 Connection in SSL Mode

When connecting to the GaussDB server through JDBC, you can enable SSL to encrypt client-server communication. This protects sensitive data transmitted over the network. This section demonstrates how to establish a database connection via API 3, DriverManager.getConnection(String url, Properties info).

Using a Certificate for Authentication

Ensure that the following operations have been performed:

- Add the **gaussdbjdbc.jar** package to your runtime environment (for example, your IDE's classpath).
- Log in to the GaussDB management console. On the **Instances** page, click the instance name to go to the **Basic Information** page. Click under **SSL** to download the root certificate or certificate bundle, and save the root certificate **ca.pem** to the client.

To configure certificates on the client to connect to a database, perform the following steps:

Step 1 Import **java.sql.Connection**, **java.sql.DriverManager**, and **java.util.Properties**.

Additionally, import other APIs and classes as needed based on your application requirements. For details, see **JDBC API Reference**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the database **sourceURL** (change *$ip*, *$port*, and *database* in the URL as required), username, and password.

Storing usernames and passwords directly in code poses significant security risks. Instead, use environment variables to securely manage credentials.

```
String sourceURL = "jdbc:gaussdb://$ip:$port/database";
Properties urlProps = new Properties();
urlProps.setProperty("user", System.getenv("EXAMPLE_USERNAME_ENV"));
urlProps.setProperty("password", System.getenv("EXAMPLE_PASSWORD_ENV"));
```

Step 3 Set the SSL attribute to **true** and configure the **ca.pem** root certificate on the client.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
urlProps.setProperty("ssl", "true");
urlProps.setProperty("sslrootcert", "ca.pem");
```

Step 4 Configure **sslmode**.

Set **sslmode** to **require**, **verify-ca**, or **verify-full**. For details about these values, see **sslmode**. Choose the appropriate option based on your requirements.

```
urlProps.setProperty("sslmode", "require"); // Set sslmode to require.
// urlProps.setProperty("sslmode", "verify-ca"); // Set sslmode to verify-ca.
// urlProps.setProperty("sslmode", "verify-full"); // Set sslmode to verify-full (verification on Linux).
```

Step 5 Load the driver.

Load the database driver **com.huawei.gaussdb.jdbc.Driver** as follows:

```
Class.forName("com.huawei.gaussdb.jdbc.Driver");
```

Step 6 Create a database connection.

Call **DriverManager.getConnection(String url, Properties info)** to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL,urlProps);
```

----End

5.3.2.2.4 Connecting to a Database in UDS Mode

The Unix domain socket (UDS) is used for data exchange between different processes on the same host. You need to add **junixsocket** to obtain the socket factory.

Prerequisites: **gaussdbjdbc.jar**, **junixsocket-core-*XXX*.jar**, **junixsocket-common-*XXX*.jar**, and **junixsocket-native-common-*XXX*.jar** have been added to your runtime environment (such as an IDE). *XXX* indicates the version number and must be consistent throughout the last three JAR packages.

To connect to a database in UDS mode, perform the following steps:

Step 1 Import java.sql.Connection, java.sql.DriverManager, and java.util.Properties.

In addition, you need to import other APIs and classes based on the actual application scenario. For details, see **JDBC API Reference**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

Step 2 Specify the database username and password, and set them as the property values of the Properties object.

Writing the username and password directly in the code creates significant security risks. It is advisable to store them in environment variables instead.

```
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Properties properties = new Properties();
properties.setProperty("user", userName);
properties.setProperty("password", password);
```

Step 3 Load the driver.

Load the database driver **com.huawei.gaussdb.jdbc.Driver**.

```
String driver = "com.huawei.gaussdb.jdbc.Driver";
Class.forName(driver);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 4 Specify the database *$ip*, *$port*, *database*, **socketFactory**, and **socketFactoryArg**.

Provide actual values for *$ip* and *$port*. Set *database* to **localhost**, **socketFactory** to **org.newsclub.net.unix.AFUNIXSocketFactory$FactoryArg**, and **socketFactoryArg** to *[path-to-the-unix-socket]* to connect to the database in UDS mode. For details about the **socketFactory** and **socketFactoryArg** parameters, see **socketFactory** and **socketFactoryArg**.

```
Connection conn = DriverManager.getConnection("jdbc:gaussdb://$ip:$port/database?
socketFactory=org.newsclub.net.unix" +
        ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=[path-to-the-unix-socket]",properties);
System.out.println("Connection Successful!");
```

NOTE

Set **socketFactoryArg** based on the actual path and ensure that it is consistent with the GUC parameter **unix_socket_directory**.

----End

5.3.2.2.5 Connection Parameter Reference

Connection parameter names are case-sensitive. **Table 5-11** lists the categories of connection parameters.

Table 5-11 Connection parameter categories

Category	Parameter
Authentication	PGDBNAME, PGHOST, PGPORT, user, password, tenantName, tenantPassword
Database connection	ApplicationName, protocolVersion, connectMode, loginTimeout, logUnclosedConnections, allowConnThreadPool
Network connection	socketFactory, socketFactoryArg, receiveBufferSize, sendBufferSize, tcpKeepAlive, connectTimeout, socketTimeout, socketTimeoutInConnecting, cancelSignalTimeout, cmNetworkTimeout
Security	ssl, sslmode, sslgmcipher, sslcert, sslkey, sslenccert, sslenckey, sslrootcert, sslpassword, sslpasswordcallback, sslfactory, sslprivatekeyfactory, sslfactoryarg, sslhostnameverifier, TLSCiphersSupperted, enable_ce, key_info, auto_sendtoken, and refreshClientEncryption
Result set	fetchsize, defaultRowFetchSize, unknownLength, uppercaseAttributeName, stripTrailingZeros, useGsClobBlobClass, enableObjectTypeConv

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Parameter
Data types	binaryTransfer, binaryTransferEnable, binaryTransferDisable, blobMode, yearIsDateType, zeroCharacterReplacement, use_boolean, setFloat, stringtype
Logging and debugging	logger, loggerLevel, loggerFile, printSqlInLog, actualErrorCodeForBatch, connectionExtraInfo, traceInterfaceClass
Compatibility	driverInfoMode, oracleCompatible, mCompatible, pgCompatible, compatibilityTags, enableObjectTypeConv, assumeMinServerVersion, mCompatibleSetDateTime, mQuoteStringCompatible
Performance	prepareThreshold, preparedStatementCacheQueries, preparedStatement-CacheSizeMiB, databaseMetadataCacheFields, databaseMetadataCache-FieldsMiB, batchMode, reWriteBatchedInserts
Availability	currentSchema, enableStandbyRead, loadBalanceHosts, autoBalance, disasterToleranceCluster, usingEip, enableAutoMigrate, priorityServers, preferredServiceGroup, preferredNodeIds, backupServiceGroup, backupNodeIds, hostRecheckSeconds, ApplicationType, sessionType, execTargetNode
Other functionalities	allowEncodingChanges, characterEncoding, enableTimeZone, autosave, cleanupSavepoints, enableStreamingQuery, preferQueryMode, allowReadOnly, parseCandidatesByDomain, primaryDomains, priorityDomains, refreshDomainResolveTime, refreshCNIpListTime, statementTimeout, executeUpdateQueryable, dbMonitor, enableGaussArrayAndStruct, getColumnsReturnNull, extendTablePrivi-lege, diffProcAndFunc

Authentication

Table 5-12 Connection parameters for authentication

Parameter	Description	Value
PGDBNAME	Specifies the database name. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file by the system.	Type: string.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
PGHOST	• Specifies the IP address of the host in non-multi-primary mode. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file by the system. Both IPv4 and IPv6 addresses are supported. If multiple CNs are configured, use colons (:) to separate their IP addresses and port numbers, and use commas (,) to separate CNs.	Type: string.
PGPORT	• Specifies the port number of the host in non-multi-primary mode. You do not need to set this parameter in the URL because it is automatically parsed from the .properties file by the system.	Type: integer.
user	Specifies the database user who creates the connection.	Type: string.
password	Specifies the password for the database user.	Type: string. **NOTICE** • To prevent the potential leakage of sensitive information, you are advised not to set this parameter in the URL. • In non-multi-primary mode, **Connecting to a Database Using API 2 (Non-Multi-Primary)** is recommended for setting this parameter. You can also set this parameter in connection properties, as described in **Connecting to a Database Using API 3 (Non-Multi-Primary)**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Database Connection

Table 5-13 Connection parameters for database connection

Parameter	Description	Value
ApplicationName	Specifies the name of the JDBC driver that is being connected. • In non-multi-primary mode, you can query the PGXC_STAT_ACTIVITY system view on CN to obtain information about connected clients. The **application_name** field indicates the JDBC driver name.	Type: string. Default value: **PostgreSQL JDBC Driver**.
protocolVersion	Specifies the version number of the connection protocol. It is used to connect to databases of different versions. **NOTE** When setting **protocolVersion**, you need to confirm the database type and encryption mode. For example, if the user wants to connect to a V5 server and set this parameter to **3**, make sure the database encryption mode is MD5. If it is not, change it to MD5. To do this, set **password_encryption_type** to **1**. Then, log in to the database as the user whose password is encrypted using the corresponding encryption mode or create a new user with MD5 encryption. You must also change the client connection mode to MD5 through the **pg_hba.conf** file. For details about the parameter **password_encryption_type**, see "Database GUC Parameters > GUC Parameters > Connection and Authentication > Security and Authentication" in *Reference*.	Type: integer. Value range: • **1**: The V1 server is connected. • **3**: The server to connect to is either earlier than V5 or is at V5 and utilizes the MD5 encryption mode. You are advised not to use this value as it could present security risks. • **350**: The server to connect to is earlier than V5. You are advised not to use this value as it could present security risks. • **351**: The server to connect to is at V5, which supports MD5, SHA256, and SM3 encryption modes. You are advised to select this value. Default value: **351**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
loginTimeout	• Specifies how long to wait for establishment of a database connection in non-multi-primary mode. When multiple IP addresses are configured in the URL, if the time for obtaining the connection exceeds this value, the connection fails and the subsequent IP addresses are not tried. **loginTimeout** includes **connectTimeout** and **socketTimeoutInConnecting**, calculated as: **loginTimeout** = (**connectTimeout** + **socketTimeoutInConnecting**) x Number of nodes.	Type: integer. Unit: s. Value range: 0–2147483647. When the value is **0** or a negative number, the timeout is disabled. Default value: **0**. Suggestion: After this parameter is set, an asynchronous thread is started each time a connection is established. If there are a large number of connections, the pressure on the client may increase. If this parameter is required, you are advised to adjust its value as follows: • In non-multi-primary mode, set the value to max(connectTimeout, socketTimeoutInConnecting) x Number of nodes. This prevents connection failures when the network is abnormal and the *n*th IP address is the IP address of the primary node. **NOTE** • For the multi-IP address configuration in non-multi-primary mode, this parameter specifies the total time for attempting to connect to the IP addresses. If the value is too small, the subsequent IP addresses may fail to be connected. For example, if three IP addresses are configured and the value of **loginTimeout** is 5s, the total time for attempting to connect to the first two IP addresses is 5s and the connection is not established, the third IP address cannot be connected. In this case, the connection fails to be established.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
	NOTE When any of the CPU, memory, and I/O load approaches 100%, the connection is slow, which may cause connection timeout. You can locate the fault as follows: 1. Log in to a physical machine with slow connections or use a management tool to query the resource load. You can run the **top** command to check the CPU usage, run the **free** command to check the memory usage, and run the **iostat** command to check the I/O load. In addition, you can check the monitoring logs in the CM Agent and the monitoring records on the database O&M platform. 2. In peak load scenarios caused by many slow queries occurring within a short time, access the pg_stat_activity view by using the port number [*Database server port* + 1]. For the slow queries, you can use the system function pg_terminate_backend(pid int) to terminate the corresponding sessions. 3. If service overloading exists for a long time (that is, there is no obvious slow query, or new queries still become slow after slow queries are terminated), reduce the service load and increase database resources.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
logUnclosedConnections	The client may leak a connection object because it does not call the close() method to close the connection object. When this parameter is set to **true**, logs are recorded when the JVM calls the finalize() method to reclaim the connection object.	Type: Boolean. Value range: • **true**: If the close() method is not called, logs are recorded when the JVM calls the finalize() method to reclaim the connection object. • **false**: If the close() method is not called, no logs are recorded when the JVM calls the finalize() method to reclaim the connection object. Default value: **false**.

Network Connection

Table 5-14 Connection parameters for network connection

Parameter	Description	Value
socketFactory	Specifies the name of the class used to create a socket connection with the server. This class must implement the javax.net.SocketFactory API and define a constructor with no parameter or a single string parameter.	Type: string.
socketFactoryArg	Specifies the optional arguments of the constructor of the socketFactory class. It is advisable not to utilize it if you are uncertain about the class of constructors.	Type: string.
receiveBufferSize	Specifies SO_RCVBUF on a connection flow.	Type: integer. Unit: byte. Value range: -1 to 2147483647. Default value: **-1**, indicating that the buffer size is not set.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
sendBufferSize	Specifies SO_SNDBUF on a connection flow.	Type: integer. Unit: byte. Value range: -1 to 2147483647. Default value: **-1**, indicating that the buffer size is not set.
tcpKeepAlive	Specifies whether to enable TCP keepalive detection.	Type: Boolean. Value range: • **true**: TCP keepalive detection is enabled. • **false**: TCP keepalive detection is disabled. Default value: **false**.
connectTimeout	Specifies the timeout period for connecting to a server OS. If the time taken to connect to a server OS exceeds the value specified, the connection is interrupted. When multiple IP addresses are configured in the URL, it indicates the timeout period for connecting to a single IP address.	Type: integer. Unit: s. Value range: 0–2147483. When the value is **0**, the timeout is disabled. Default value: **0**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
socketTimeout	Specifies the timeout period for a socket read operation. If the time taken to read data streams from a server exceeds the value specified, the connection is closed. You are advised to set this parameter. Otherwise, when the database process is abnormal or the server system is restarted but the socket is not closed, the client may wait for a long time. You are advised to set this parameter based on the SQL execution time acceptable to services. **NOTE** When the timeout is triggered on the JDBC and the connection is closed, the running services delivered by the JDBC to the database are forcibly terminated. The capability of forcibly terminating a service is controlled by the GUC parameter **check_disconnect_query**. It is supported when the parameter is set to **on** and not supported when the parameter is set to **off**.	Type: integer. Unit: s. Value range: 0–2147483. When the value is **0** or a negative number, the timeout is disabled. Default value: **0**.
socketTimeoutInConnecting	Specifies the timeout period for a socket read operation during connection establishment. If the time of reading data streams from the server exceeds the threshold, it attempts to search for the next node for connection.	Type: integer. Unit: s. Value range: 0–2147483. When the value is **0** or a negative number, the timeout is disabled. Default value: **5**.
cancelSignalTimeout	Specifies the timeout period for the connect timeout and socket timeout of the cancel command. If the cancel command does not respond within the specified time, the connection is interrupted to reduce the occupation of client resources.	Type: integer. Unit: s. Value range: 0–2147483. When the value is **0** or a negative number, the timeout is disabled. Default value: **10**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Security

Table 5-15 Connection parameters for security

Parameter	Description	Value
ssl	Specifies whether to connect to the database in SSL mode. If set to **true**, SSL connections can be established using either the NonValidatingFactory channel or a certificate.	Type: Boolean. Value range: • **true**: The database is connected in SSL mode. • **false**: The database is not connected in SSL mode. Default value: **NULL** (equivalent to **false**).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
sslmode	Specifies the SSL authentication mode.	Type: string. Value range: **disable**, **allow**, **prefer**, **require**, **verify-ca**, and **verify-full**. • **disable**: SSL connections are not used. • **allow**: SSL connections are used when required by the database server. However, this mode does not verify the authenticity of the database server. • **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. • **require**: The system only attempts to set up an SSL connection. It neither checks whether the server certificate is issued by a trusted CA, nor checks whether the host name of the server is the same as that in the certificate. • **verify-ca**: The system attempts to set up an SSL connection and checks whether the server certificate is issued by a trusted CA. • **verify-full**: The system will only attempt to establish an SSL connection. It will verify whether the server holds a certificate issued by a trusted CA and whether the host name of the server is the same as that in the certificate. Default value: **NULL** (equivalent to **require**).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
sslgmcipher	Specifies the encryption algorithm suite used by SM-based TLS.	Type: string. Value range: • **ECC_SM4_SM3**: SM-based one-way and two-way authentication modes are supported. • **ECDHE_SM4_SM3**: Only SM-based two-way authentication is supported. Default value: **NULL**.
sslcert	Specifies the complete path of the certificate file. The certificate type is **End Entity**.	Type: string. Default value: **NULL** (indicating that the client certificate file in the user's root directory is read).
sslkey	Specifies the complete path of the key file. You need to convert the key to the DER format before using it. `openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt`	Type: string. Default value: **NULL** (indicating that the key file in the user's root directory is read).
sslenccert	Specifies the complete path of the encryption certificate for the SM-based TLS client. The certificate type is **End Entity**.	Type: string. Default value: **NULL** (indicating that the SM-based TLS client encryption certificate in the user's root directory is read).
sslenckey	Specifies the complete path of the SM-based TLS key file. You need to convert the key to the DER format before using it. `openssl pkcs8 -topk8 -outform DER -in client_enc.key -out client_enc.key.pk8 -nocrypt`	Type: string. Default value: **NULL** (indicating that the SM-based TLS key file in the user's root directory is read).
sslrootcert	Specifies the name of the SSL root certificate. The root certificate type is **CA**.	Type: string. Default value: **NULL** (indicating that the SSL root certificate in the user's root directory is read).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
sslpassword	Specifies the SSL password. This parameter is provided for the ConsoleCallbackHandler component.	Type: string. Default value: **NULL**. **NOTICE** • To prevent the potential leakage of sensitive information, do not set this parameter in the URL. • You are advised to set this parameter in connection properties, as described in **Connecting to a Database Using API 3 (Non-Multi-Primary)**.
sslpasswordcallback	Specifies the full class name of the SSL password provider.	Type: string. Default value: **NULL** (indicating that the functionality is disabled).
sslfactory	Specifies the class name used by SSLSocketFactory to establish an SSL connection.	Type: string. Default value: **NULL** (indicating that the functionality is disabled).
sslprivatekeyfactory	Specifies the fully qualified name of the implementation class of the com.huawei.gaussdb.jdbc.ssl.PrivateKeyFactory API that implements the private key decryption method. If this parameter is not specified, the system will first attempt to use the default JDK private key decryption algorithm. If the decryption fails, **com.huawei.gaussdb.jdbc.ssl.BouncyCastlePrivateKeyFactory** will be used. In this case, you need to provide the **bcpkix-jdk15on.jar** package (version 1.65 or later is recommended).	Type: string. Default value: **NULL** (indicating that the system will attempt to use the default JDK private key decryption algorithm).
sslfactoryarg	Specifies the optional parameter of the constructor function of the sslfactory class. (This parameter is not recommended.)	Type: string. Default value: **NULL**.
sslhostnameverifier	Specifies the class name of the host name verifier. This parameter implements the javax.net.ssl.HostnameVerifier API.	Type: string. Default value: **com.huawei.gaussdb.jdbc.ssl.PGjdbcHostnameVerifier**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
TLSCiphersSupperted	Specifies the list of supported TLS cipher suites. **NOTE** Setting an unsupported TLS cipher suite will result in an error being reported.	Type: string. Default value: **TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**.
enable_ce	Specifies the mode in which encrypted equality query is enabled.	Type: integer. Value range: • **1** or **3**: JDBC supports the basic functionality of encrypted equality query. • Other values: The basic functionality of encrypted equality query is not supported. Default value: **NULL** (indicating that the basic functionality of encrypted equality query is not supported).
key_info	Used together with **enable_ce** to configure parameters for accessing the external key management service. This parameter takes effect only in the encrypted database environment.	Type: string. Default value: **NULL**. **NOTICE** • To prevent the potential leakage of sensitive information, you are advised not to set this parameter in the URL. • You are advised to set this parameter in connection properties, as described in **Connecting to a Database Using API 3 (Non-Multi-Primary)**.
auto_sendtoken	Used together with **enable_ce** and takes effect only when **enable_ce** of the encrypted database is set to **3**. After this parameter is enabled, the JDBC client automatically performs the same key transmission operation as the setClientInfo("send_token") functionality after a connection is established.	Type: Boolean. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
refreshClientEncryption	Specifies whether the encrypted database supports cache update on the client.	Type: string. Value range: • **1** or **NULL**: The encrypted database supports cache update on the client. • Other character strings: The encrypted database does not support cache update on the client. Default value: **NULL**.

Result Set

Table 5-16 Connection parameters for a result set

Parameter	Description	Value
fetchsize	Specifies the number of rows fetched from ResultSet each time. This parameter is equivalent to the **defaultRowFetchSize** parameter.	Type: integer. Value range: 0–2147483647. The value **0** or a negative number indicates that all rows are fetched from ResultSet each time. Default value: **0**. Suggestion: Set this parameter based on the actual amount of data to be queried and the memory usage of the client. When setting **fetchsize**, disable the automatic commit mode (conn.setAutoCommit(false);). Otherwise, **fetchsize** will not take effect.
defaultRowFetchSize	Specifies the number of rows fetched from ResultSet each time. Limiting the number of rows read each time in a database access request can avoid unnecessary memory consumption, thereby avoiding the out of memory exception. **NOTICE** If both **fetchsize** and **defaultRowFetchSize** are set, the value of **fetchsize** is used.	Type: integer. Value range: 0–2147483647. The value **0** indicates that all rows are fetched from ResultSet each time. Default value: **0**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
unknownLength	Specifies the data size of some GaussDB data types (such as TEXT) whose size is unknown and not defined when the data are returned by methods such as java.sql.ResultSetMetaData#getColumnDisplaySize and java.sql.ResultSetMetaData#getPrecision.	Type: integer. Value range: 0–2147483647. Default value: **2147483647**.
uppercaseAttributeName	Specifies whether to convert the query result of the API for obtaining metadata to uppercase or lowercase letters. The application scenario is as follows: All metadata stored in the database is in lowercase, but they must be used as the input and output parameters in uppercase. For details about the involved APIs, see **java.sql.DatabaseMetaData** and **java.sql.ResultSetMetaData**. **NOTE** After the **uppercaseAttributeName** parameter is enabled, if the database contains metadata with a mixture of uppercase and lowercase letters, only the metadata in lowercase letters can be queried and output in uppercase letters. Before using this parameter, ensure that the metadata is stored in lowercase letters to prevent data errors.	Type: string. Value range: • **true** or **upper**: Metadata is converted to uppercase letters. • **false**: The configuration of GUC parameter **uppercase_attribute_name** is used. • **lower**: Metadata is converted to lowercase letters. Default value: **false**.
stripTrailingZeros	Specifies whether trailing 0s of the numeric type are removed. It is valid only for java.sql.ResultSet#getObject(int columnIndex).	Type: Boolean. Value range: • **true**: Trailing 0s of the numeric type are removed. • **false**: Trailing 0s of the numeric type are not removed. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
useGsClobBlobClass	Specifies the type of the object returned when the java.sql.ResultSet#getObject method is used to obtain the CLOB and BLOB columns, and the type name returned when the java.sql.ResultSetMetaData#getColumnClassName method is used to obtain the CLOB and BLOB columns.	Type: Boolean. Value range: • **true**: When obtaining CLOB columns, the java.sql.ResultSet#getObject method returns an object of the PGClob type. When obtaining BLOB columns, it returns an object of the PGBlob type. When the java.sql.ResultSetMetaData#getColumnClassName method is used to obtain the type name of a CLOB column, java.sql.Clob is returned. When it is used to obtain the type name of a BLOB column, java.sql.Blob is returned. • **false**: When obtaining CLOB columns, the java.sql.ResultSet#getObject method returns an object of the PGClob type. When obtaining BLOB columns, it returns an object of the byte[] type. When the java.sql.ResultSetMetaData#getColumnClassName method is used to obtain the type name of a CLOB column, java.sql.CLOB is returned. When it is used to obtain the type name of a BLOB column, java.sql.BLOB is returned. Default value: **false**.
enableObjectTypeConv	Specifies whether the java.sql.ResultSet#getObject(int, Class<?>) API allows compatible type conversion. Conversion between numeric types, between string types, and between date/time types is allowed.	Type: Boolean. Value range: • **true**: Compatible type conversion is allowed. • **false**: Compatible type conversion is not allowed. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Types

Table 5-17 Connection parameters for data types

Parameter	Description	Value
binaryTransfer	Specifies whether data is sent and received in binary format.	Type: Boolean. Value range: • **true**: Data is transferred in binary format. • **false**: Data is transferred in text format. Default value: **false**.
binaryTransferEnable	Specifies types for which binary transmission is enabled. Separate them with commas (,), for example, **binaryTransferEnable=INT4_ARRAY,INT8_ARRAY**. You can use the OID or name for configuration. For example, if the name is BLOB and the OID is 88, set **binaryTransferEnable** to **BLOB** or **88**. **NOTICE** If **binaryTransferDisable** and **binaryTransferEnable** have the same value, the disabling function is used.	Type: string. Default value: **""** (an empty string).
binaryTransferDisable	Specifies types for which binary transmission is disabled. The values are separated by commas (,). You can use the OID or name for configuration.	Type: string. Default value: **""** (an empty string).
blobMode	Specifies the setBinaryStream() method to assign values to different types of data.	Type: string. Value range: • **on**: Values are assigned to BLOB data. • **off**: Values are assigned to bytea data. Default value: **on**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
yearIsDateType	Specifies whether the returned data object is displayed in the date type when the java.sql.ResultSet#getObject and java.sql.ResultSet#getString APIs are used to obtain columns of the year type.	Type: Boolean. Value range: • **true**: When obtaining columns of the year type, the java.sql.ResultSet#getObject method returns an object of the date type. When obtaining year data, the java.sql.ResultSet#getString method returns a date string in yyyy-mm-dd format. If the java.sql.ResultSet#getInt/ getLong/getShort method is used, it returns a year number. • **false**: When obtaining columns of the year type, the java.sql.ResultSet#getObject method returns an object of the integer type. When obtaining year data, the java.sql.ResultSet#getString method returns a year string. If the java.sql.ResultSet#getInt/ getLong/getShort method is used, it returns a year number. Default value: **true**.
use_boolean	Sets the OID type bound to the setBoolean() method in extended mode.	Type: Boolean. Value range: • **true**: The Boolean type is bound. • **false**: The int2 (Smallint) type is bound. Default value: **false**.
setFloat	Specifies whether the OID passed to the database is an OID of float4 when **setFloat** is called or **setObject** is set to **float**. This parameter does not apply to databases in M-compatible mode.	Type: Boolean. Value range: • **true**: The OID passed to the database is of the float4 type when **setFloat** is called or **setObject** is set to **float**. • **false**: The OID passed to the database is of the float8 type when **setFloat** is called or **setObject** is set to **float**. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
stringtype	Specifies the type of the parameter passed to the database when the java.sql.PreparedStatement#setString method is called.	Type: string. Value range: • **varchar**: Parameters are sent to the server as varchar parameters. • **unspecified**: No type is specified when parameters are sent to the server. The server will attempt to infer an appropriate type. Default value: **NULL** (equivalent to **varchar**).

Logging and Debugging

Table 5-18 Connection parameters for logging and debugging

Parameter	Description	Value
logger	Specifies the log output framework used by the JDBC driver. The JDBC driver supports the log output framework used for interconnecting with user applications.	Type: string. Value range: • **Slf4jLogger**: Slf4j-API-based log framework. Currently, only the Slf4j-API-based third-party log output framework is supported. For details, see **Log Management**. • **JdkLogger**: Java built-in log framework. If this parameter is not explicitly configured, the driver uses **JdkLogger** by default.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
loggerLevel	Specifies the logging level.	Type: string. Value range: The following levels are supported, which are case-insensitive. ● **OFF**: disables all log output. ● **INFO**: records basic running information. ● **DEBUG**: records debugging information (more than that of **INFO**). ● **TRACE**: records the most detailed debugging information (highest level). Default value: **OFF**.
loggerFile	Specifies the output path of monitoring logs when the connection monitoring functionality is enabled. This parameter is valid only in the connection monitoring scenario.	Type: string. Default value: **NULL** (indicating that log files are generated in the directory where the client program is running).
printSqlInLog	Specifies whether to output SQL statements in exception information or logs.	Type: Boolean. Value range: ● **true**: SQL statement output is enabled. ● **false**: SQL statement output is disabled. Default value: **true**.
actualErrorCodeForBatch	Specifies whether to return the actual error code or fixed error code **0** when an error is reported during the execution of the executeBatch API.	Type: Boolean. Value range: ● **true**: The actual error code is returned. ● **false**: The fixed error code 0 is returned. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
connectionExtraInfo	Specifies whether to report the driver information to the database.	Type: Boolean. Value range: • **true**: The driver deployment path, process owner, and URL connection configuration are reported to the database. The information is recorded in the **connection_info** parameter. **NOTE** • In non-multi-primary mode, you can view the driver information using PG_STAT_ACTIVITY and PGXC_STAT_ACTIVITY. • **false**: Information is not reported. Default value: **false**.
traceInterfaceClass	Specifies a fully qualified name for the class written to implement the com.huawei.gaussdb.jdbc.log.Tracer API and the method for obtaining **traceId**.	Type: string. Default value: **NULL** (indicating that the functionality is disabled).

Compatibility

Table 5-19 Connection parameters for compatibility

Parameter	Description	Value
driverInfoMode	Specifies the output mode of the driver description information.	Type: string. Value range: • **postgresql**: The driver description related to PostgreSQL is displayed. • **gaussdb**: The driver description related to GaussDB is displayed. Default value: **postgresql**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
oracleCompatible	Specifies whether to enable Oracle-compatible features on the driver side.	Type: string. Value range: • **true**: All Oracle compatibility tags are enabled on the driver side. (After the driver is upgraded, the tags added in the new version are automatically included.) • **false**: All Oracle-compatible tags are disabled on the driver side. • **tag**: Some Oracle-compatible features are enabled on the driver side, with one tag per feature. If multiple tags are configured, separate them with commas (,), for example, "tag1,tag2,tag3". Currently, the following tags can be configured: – **getProcedureColumns**: The DatabaseMetaData#getProcedureColumns method is compatible with the Oracle database behavior. When no catalog or schema is specified, this tag is not used as a filter criterion by default. – **getCallableStatementResults**: The getLong, getInt, getShort, and getByte methods of the CallableStatement API are compatible with the Oracle database behavior. When the registered output parameter type is java.sql.Types#NUMERIC, you can call the getLong, getInt, getShort, and getByte methods of CallableStatement to receive the value of **out**. The SQLException message is displayed only when the value of **out** exceeds the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
		value range of the Java numeric data type. – **batchInsertAffectedRows**: After the **reWriteBatchedInserts** parameter is enabled, the Statement#executeBatch method is compatible with the Oracle database behavior. – **setDateTime**: When parameters of the date, time, and timestamp types are bound in Prepare-Bind-Execute (PBE) mode, the OID is specified and delivered to the database. – **blob**: When enabled, BLOB APIs are compatible with the Oracle database behavior. – **clob**: When enabled, CLOB APIs are compatible with the Oracle database behavior. – **functionsCompatible**: The following methods are compatible with the Oracle database behavior: – ResultSet#wasNull, getBoolean, insertRow – DatabaseMetaData#getProcedureTerm – Blob#length, getBytes, position, setBytes, truncate, free – Clob#length, getSubString, getCharacterStream, getAsciiStream, position, setString, truncate, free – CallableStatement. For details, see **java.sql.CallableStatement**. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
mCompatible	Specifies whether to enable MySQL-compatible features on the driver side.	Type: string. Value range: • **true**: All MySQL-compatible tags are enabled on the driver side. (After the driver is upgraded, the tags added in the new version are automatically included.) • **false**: All MySQL-compatible tags are disabled on the driver side. • **tag**: Some MySQL-compatible features are enabled on the driver side, with one tag per feature. If multiple tags are configured, separate them with commas (,), for example, "tag1,tag2,tag3". Currently, the following tags can be configured: – **DatetimeAsTimeStamp-Type**: An object of the Timestamp type is returned when the java.sql.ResultSet#getObject method is used to obtain a column of the Datetime data type. – **blob**: When enabled, BLOB APIs are compatible with the MySQL database behavior. – **clob**: When enabled, CLOB APIs are compatible with the MySQL database behavior. – **functionsCompatible**: The following methods are compatible with the MySQL database behavior: – ResultSet#wasNull – DatabaseMetaData#getProcedureTerm – ResultSetMetaData#getColumnName – Blob#length, getBytes, setBytes, truncate, free

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
		– Clob#length, getSubString, getCharacterStream, getAsciiStream, position, setString, truncate, free Default value: **false**.
mCompatibleSetDateTime	Binds parameters of the date, time, and timestamp types and specifies the OID as a specific value or **0** to deliver it to the database when a MySQL-compatible database is connected.	Type: Boolean. Value range: ● **true**: The parameters of the date, time, and timestamp types are bound, and the specified OID is delivered to the database, which is compatible with the PBE mode (useServerPrepStmts=true&cachePrepStmts=true) of the MySQL database. ● **false**: The parameters of the date, time, and timestamp types are bound, and the OID is set to **0** and delivered to the database. The server automatically infers the type, which is compatible with the default query mode of the MySQL database. Default value: **false**.
pgCompatible	Specifies whether the SQL syntax parsing mode of JDBC is compatible with PG.	Type: Boolean. Value range: ● **true**: Syntax parsing is compatible with PG. A statement can contain multiple SQL statements separated by semicolons (;). When enabled, SQL statements containing semicolons (;) cannot be executed. ● **false**: Syntax parsing operates in the default mode, which is incompatible with PG. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
compatibilityTags	Restores some features of the driver to an earlier version to ensure backward compatibility. You can configure corresponding tags. Each tag indicates that the corresponding backward compatibility feature is enabled. Different tags are separated by commas (,). Currently, the following tag is supported: **typeMapInitNull**: compatible with the behavior of earlier driver versions. **NOTE** Generally, this parameter is not recommended unless there is an obvious backward compatibility issue.	Type: string. Default value: **NULL**.
enableObjectTypeConv	Specifies whether the java.sql.ResultSet#getObject(int, Class<?>) method allows compatible type conversion. Conversion between numeric types, between string types, and between date/time types is allowed.	Type: Boolean. Value range: • **true**: Compatible type conversion is allowed. • **false**: Compatible type conversion is not allowed. Default value: **false**.
assumeMinServerVersion (Deprecated)	Specifies the minimum version of the connected server. For example, if the value of **assumeMinServerVersion** is 9.0 or later, the connection establishment process can be optimized. The client will no longer send requests to set the float precision to 3 (the default precision level is 2).	Type: string. Default value: **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
mQuoteStringCompatible	Specifies whether the java.sql.DatabaseMetaData#getIdentifierQuoteString() method MySQL-compatible.	Type: Boolean. Value range: • **true**: MySQL-compatible. If the **sql_mode** parameter of the database contains the **ANSI_QUOTES** option, the method returns double quotation marks. Otherwise, the method returns backquote. • **false**: not MySQL-compatible. The method returns double quotation marks. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Performance

Table 5-20 Connection parameters for performance

Parameter	Description	Value

prepareThreshold	Specifies the threshold for the number of times that a PreparedStatement object is executed. When the threshold is reached, the parsed statements on the server will be reused.
	NOTE
	● When this functionality is enabled, you are not advised to use the prepareStatement method to execute statements involving passwords (for example, CREATE USER user_name WITH PASSWORD '********';). This is because the database caches SQL statements when the number of execution times reaches the value of prepareThreshold. For security purposes, the database does not cache passwords. When the prepareStatement method is executed again, the error message "Password must contain at least 8 characters." is displayed.
	● Setting **prepareThreshold** to **0** or **1** is not recommended.
	  ● If this parameter is set to **0**, SQL statements need to be parsed again for each execution, which deteriorates database performance.
	  ● If this parameter is set to **1**, the execution plan is cached even if the statements are executed only once, causing unnecessary memory consumption.
		Type: integer.
		Value range: 0–2147483647. When the value is **0** or a negative number, the functionality is disabled.
		Default value: **5**. After the number of times that the same PreparedStatement object is executed reaches 5, the database does not send a parse message to the server for repeated parsing. Instead, the database reuses the parsed statements on the server.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
preparedStatementCacheQueries	When the number of queries created using prepareStatement() in each connection exceeds a specified threshold, the least recently used (LRU) policy is used to evict the query cache.	Type: integer. Value range: 0–2147483647. When the value is **0** or a negative number, the cache is disabled. Default value: **256**. When the number of queries created using prepareStatement() in each connection exceeds 256, the least recently used (LRU) policy is used to evict the query cache. Suggestion: Adjust this parameter based on service requirements. This parameter must be used together with the **prepareThreshold** parameter.
preparedStatementCacheSizeMiB	When the cache space occupied by queries created using prepareStatement() in each connection exceeds a specified threshold, the least recently used (LRU) policy is used to evict the query cache.	Type: integer. Unit: MB. Value range: 0–2147483647. When the value is **0** or a negative number, the cache is disabled. Default value: **5**. When the cache space occupied by queries created using prepareStatement() in each connection exceeds 5 MB, the least recently used (LRU) policy is used to evict the query cache.
databaseMetadataCacheFields	Specifies the maximum number of database metadata fields that can be cached in each connection.	Type: integer. Value range: 0–2147483647. When the value is **0** or a negative number, the cache is disabled. Default value: **65536**.
databaseMetadataCacheFieldsMiB	Specifies the maximum cache space occupied by database metadata fields that can be cached in each connection.	Type: integer. Unit: MB. Value range: 0–2147483647. When the value is **0** or a negative number, the cache is disabled. Default value: **5**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
batchMode	Specifies whether to use the batch mode. **NOTE** If **batchMode** is set to **on** and batch insert/update/delete operations are performed, the data type of each column is the same as that specified by the first data record. If the data types are mixed, an error may be reported or the inserted data may be abnormal.	Type: string. Value range: • **on**: The batch mode is enabled to improve the batch insert/update/delete performance. After the batch insert/update/delete operation is successfully executed, the return result is [count, 0, 0...0]. The first element in the array indicates the total number of rows inserted/updated/deleted in batches. • **off**: The batch mode is disabled. After the insert/update/delete operation is successfully executed, the number of rows affected by a single SQL statement corresponds to each element in the returned result array. Default value: **on**.
reWriteBatchedInserts	Specifies whether to rewrite SQL statements during batch insertion. When this parameter is used, **batchMode** must be set to **off**.	Type: Boolean. Value range: • **true**: During batch insertion, insert statements can be combined into one. `INSERT INTO TABLE_NAME VALUES (values1, ..., valuesN), ..., (values1, ..., valuesN);` • **false**: SQL statements are not rewritten during batch insertion. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Availability

Table 5-21 Connection parameters for availability

Parameter	Description	Value
currentSchema	Specifies the schema of the current connection. If the schema name contains special characters except letters, digits, and underscores (_), you need to enclose the schema name in quotation marks. Note that the schema name is case-sensitive after quotation marks are added. If multiple schemas need to be configured, separate them with commas (,). Example: **currentSchema=schema_a,"schema-b","schema/c"**	Type: string. Default value: **NULL** (indicating that the username used for the connection is used as the default schema).
enableStandbyRead	Specifies whether to enable read on standby mode.	Type: Boolean. Value range: ● **true**: enabled. ● **false**: disabled. Default value: **false**.
loadBalanceHosts	Specifies whether to enable the load balancing functionality. **NOTE** ● This parameter cannot be used together with the **autoBalance** parameter.	Type: Boolean. Value range: ● **true**: The shuffle algorithm is used to randomly select a host from the candidates to establish a connection. ● **false**: Multiple hosts specified in the URL are connected in sequence. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value

autoBalance	• Specifies the load balancing policy in non-multi-primary mode.
	NOTICE
	• Load balancing is based on the connection level rather than the transaction level. If connections are persistent and the load on the connections is unbalanced, the load on the CNs may be unbalanced.
	• This API is used only in distributed scenarios.
	• After this parameter is configured, the IP address and port number in the **pgxc_node** table are used for subsequent connections to the database, except for the first connection. For details, see the notice of the **usingEip** parameter.
	• This parameter cannot be used together with the **loadBalanceHosts** parameter.
	• If the load balancing functionality is enabled when the standby cluster for DR is connected, query requests are sent to the primary cluster. As a result, the query performance is affected.
	• When load balancing is enabled, the floating IP address or data IP address can be configured in the URL. If the floating IP address is configured, the system obtains the corresponding data IP address based on the floating IP address and performs load balancing based on the obtained data IP address. Therefore, when configuring the floating IP address or data IP address in the URL, ensure that the network connection of the data IP address is normal. Otherwise, the load balancing functionality is abnormal.
		• In non-multi-primary mode:
		Type: string.
		Value range:
		– **true**, **balance**, or **roundrobin**: JDBC load balancing functionality is enabled to balance multiple connections from an application to available CNs in the database cluster.
		Example:
		`jdbc:gaussdb://host1:port1,host2:port2/database?autoBalance=true`
		JDBC periodically updates the list of available CNs in the entire cluster (data IP addresses of hosts in the CN list). The update interval can be configured using the **refreshCNIpListTime** parameter. The default value is 10 seconds. Here is an example list:
		`host1:port1,host2:port2,host3:port3,host4:port4`
		When load balancing is enabled, **host1** and **host2** ensure high availability only when they are connected for the first time. Then, the driver selects available CNs from **host1**, **host2**, **host3**, and **host4** in sequence to update the available CN list. New connections will use the round robin algorithm to select CNs from **host1**, **host2**, **host3**, and **host4** for connection.
		– **priority***n*: JDBC priority-based load balancing is enabled to balance multiple connections from an application to the first *n* available CNs configured in the URL. If all *n* CNs are unavailable, connections will be randomly allocated to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value

		other available CNs in the database cluster. *n* is a positive integer and is less than the number of CNs configured in the URL. When **parseCandidatesByDomain** is set to **true**, the priority configuration of **priority***n* does not take effect. In this scenario, the priority is specified by **priorityDomains**. When **priority***n* is used for load balancing, the host cannot use an IPv6 address. In the scenario where multiple processes establish connections, this policy may cause load imbalance.
		Example:
		`jdbc:gaussdb://host1:port1,host2:port2,host3:port3,host4:port4/database?autoBalance=priority2`
		JDBC periodically obtains the list of available CNs in the entire cluster. Here is an example list:
		`host1:port1,host2:port2,host3:port3,host4:port4,host5:port5,host6:port6`
		host1 and **host2** are in AZ 1, and **host3** and **host4** are in AZ 2. The driver preferentially selects **host1** and **host2** for load balancing. If both **host1** and **host2** are unavailable, the driver randomly selects CNs from **host3**, **host4**, **host5**, and **host6** for connection.
		– If this parameter is set to **shuffle**, JDBC random load balancing is enabled to randomly and evenly allocate multiple connections of the application to available CNs in the database cluster.
		Example:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
		jdbc:gaussdb://host1:port1,host2:port2,host3:port3/database?autoBalance=shuffle JDBC periodically obtains the list of available CNs in the entire cluster. Here is an example list: host1:port1,host2:port2,host3:port3,host4:port4 **host1**, **host2**, and **host3** ensure high availability only when they are connected for the first time. Then, they use the shuffle algorithm to randomly select a CN from the updated CN list for connection. – **specified**: JDBC load balancing is enabled to establish connections only on nodes specified in the user URL. Example: jdbc:gaussdb://host1:port1,host2:port2,host3:port3,host4:port4/database?autoBalance=specified&priorityServers=2 JDBC identifies **host1** and **host2** as the nodes for establishing connections in the primary cluster, and identifies **host3** and **host4** as the nodes for establishing connections in the standby cluster. When a connection is set up, JDBC evenly distributes the connections to **host1** and **host2**. If all the nodes (**host1** and **host2**) in the primary cluster are faulty, JDBC does not attempt to connect to the remaining nodes. After a switchover, JDBC evenly distributes connections to **host3** and **host4**. If all nodes (**host3** and **host4**) in the standby cluster are faulty, JDBC does not attempt to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value

		connect to the remaining nodes.
		– **shuffle***n*: JDBC random load balancing is preferred on the specified nodes to randomly balance multiple connections from an application to the first *n* available CNs configured in the URL. If all *n* CNs are unavailable, connections will be randomly allocated to other available CNs in the database cluster. *n* is a positive integer and is less than or equal to the number of CNs configured in the URL.
		When **parseCandidatesBy-Domain** is set to **true**, the priority configuration of **shuffle***n* does not take effect. In this scenario, the priority is specified by **priorityDomains**. When **shuffle***n* is used for load balancing, the host cannot use an IPv6 address. In the scenario where multiple processes establish connections, this policy can be used to implement load balancing for connections to specified preferred nodes.
		Example:
		`jdbc:gaussdb://host1:port1,host2:port2,host3:port3,host4:port4/database?autoBalance=shuffle2`
		JDBC periodically obtains the list of available CNs in the entire cluster. Here is an example list:
		`host1:port1,host2:port2,host3:port3,host4:port4,host5:port5,host6:port6`
		The driver preferentially selects **host1** and **host2** for random load balancing. If both **host1** and **host2** are unavailable, the driver randomly selects CNs from

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
		host3, **host4**, **host5**, and **host6** for connection. – If this parameter is set to **false**, the JDBC load balancing and priority-based load balancing functionalities are disabled. Default value: **false**.
disasterToleranceCluster	Specifies whether to connect to the primary cluster or the standby cluster for DR when the load balancing functionality is enabled and only standby cluster nodes are configured in the URL for the JDBC to connect to the database. This parameter cannot be used together with the **priorityServers** parameter. **NOTE** • When this parameter is configured, only the primary cluster or DR cluster can be configured in the connection string.	Type: Boolean. Value range: • **true**: Load balancing is conducted among nodes in the standby cluster. • **false**: Load balancing is conducted among nodes in the primary cluster. Default value: **false**.
usingEip	Specifies whether to use service IP addresses for load balancing. **NOTICE** • This parameter must be used together with the **autoBalance** parameter. • If **usingEip** is set to **true**, service IP addresses are used to establish connections with the database except for the first connection (corresponding to the **node_host1** and **node_port1** columns in the **pgxc_node** table). If **usingEip** is set to **false**, data IP addresses are used to establish connections with the database except for the first connection (corresponding to the **node_host** and **node_port** columns in the **pgxc_node** table).	Type: Boolean. Value range: • **true**: Service IP addresses are used to implement load balancing. • **false**: Data IP addresses are used to implement load balancing. Default value: **true**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
priorityServers	Specifies the number of nodes that are preferentially connected in the URL. It is used in streaming DR scenarios. This parameter cannot be used together with the **disasterToleranceCluster** parameter. Example: `jdbc:gaussdb://host1:port1,host2:port2,host3:port3,host4:port4/database?priorityServers=2` That is, **host1** and **host2** are primary cluster nodes, and **host3** and **host4** are standby cluster nodes for DR.	Type: integer. Value range: (0,*number of CNs configured in the URL*). Default value: **NULL** (indicating that the functionality is disabled).
hostRecheckSeconds	After JDBC attempts to connect to a host, the host status (indicating connection success or failure, and whether it is a primary or standby node) is recorded. This recorded host status remains trusted for the specified **hostRecheckSeconds**. Once this period expires, the recorded host status is no longer valid.	Type: integer. Unit: s. Value range: 0–2147483. Default value: **10**.
ApplicationType	Specifies whether to enable distributed write and query. This parameter is valid only when the database is in GTM-free mode.	Type: string. Value range: • **not_perfect_sharding_type**: enables distributed write and query. • **perfect_sharding_type**: disables distributed write and query by default. Distributed write and query can be performed only after /* multinode */ is added to the SQL statement. Default value: **not_perfect_sharding_type**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
sessionType	Specifies the type of the current session and generates the corresponding execution plan.	Type: string. Value range: • **normal**: indicates a common session type. The execution plan is generated based on the original logic of the database. • **singleNode**: indicates a single-node session type. The read and write operations can be pushed down. In single-node session mode, a CN sends a one-to-one session to a specified target DN. This parameter must be used together with **execTargetNode**. Set **execTargetNode** to the name of the target DN. JDBC preferentially connects to the local CN corresponding to the target DN. • **globalReadOnly**: indicates a cross-node read-only session type. In this mode, distributed plans are generated normally, but distributed write operations are prohibited. Default value: **normal**.
execTargetNode	Specifies a DN in single-node session mode. This parameter must be used together with **sessionType** and takes effect only when **sessionType** is set to **singleNode**. **NOTE** • If **sessionType** is set to **singleNode** but **execTargetNode** is not specified, the effect is the same as that when **sessionType** is set to **normal**.	Type: string. **Value range**: names of all DNs in the current cluster. Default value: **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Other Functionalities

Table 5-22 Connection parameters for other functionalities

Parameter	Description	Value
allowEncodingChanges	Specifies whether the character set type can be modified. This parameter is usually used together with the **characterEncoding** parameter and is connected by an ampersand (&). An example format is as follows: `allowEncodingChanges=true&characterEncoding=UTF8`	Type: Boolean. Value range: • **true**: The character set type can be changed. • **false**: The character set type cannot be changed. Default value: **false**.
characterEncoding	Specifies the character set type. This parameter must be used together with the **allowEncodingChanges** parameter. If **allowEncodingChanges** is set to **false**, the value of **characterEncoding** can only be **UTF8**. If **allowEncodingChanges** is set to **true**, **characterEncoding** can be set to any supported value. **NOTICE** • When configuring GB18030_2022, ensure that the database supports this character set. • JDK 8 1.8.0_381 and later versions support GB18030 2022. JDK 8 earlier than 1.8.0_381 supports only GB18030 2000. If JDK 8 of an earlier version is used and GB18030_2022 is configured, the character set used by JDBC will be degraded to GB18030. • JDK 17 17.0.8 and later versions support GB18030 2022. JDK 17 earlier than 17.0.8 supports only GB18030 2000. If JDK 17 of an earlier version is used and GB18030_2022 is configured, the character set used by JDBC will be degraded to GB18030.	Type: string. Value range: **UTF8**, **GBK**, **LATIN1**, **GB18030**, **GB18030_2022**, and **ZHS16GBK**. Default value: **UTF8**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
enableTimeZone	Specifies whether to enable the time zone setting on the client.	Type: Boolean. Value range: • **true**: The time zone setting on the client is enabled. The database time zone is specified based on the JVM time zone. • **false**: The time zone setting on the client is disabled. Instead, the database time zone is used. Default value: **true**.
autosave	Specifies the driver's policy to handle SQL statement execution failures within a transaction.	Type: string. Value range: • **always**: The JDBC driver sets a savepoint before executing each SQL statement and rolls back to the savepoint if the query fails. • **never**: No savepoint is set. • **conservative**: A savepoint is set before each SQL statement is executed. The system rolls back and retries only when a specific exception occurs, such as "statement *XXX* is invalid." Default value: **never**.
cleanupSavepoints	Specifies whether to automatically release the savepoints created by JDBC (when autosave is set to **always** or **conservative**).	Type: Boolean. Value range: • **true**: The savepoints created by JDBC are automatically released. • **false**: This functionality is disabled. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
enableStreamingQuery	Specifies whether to enable the streaming read functionality. **NOTE** Streaming read: When data is read, the server sends all data to the socket buffer of the client. When the buffer is full, the sending pauses. When there is free space in the buffer, the server continues to send data to the buffer. At the same time, the JVM reads data from the buffer line by line. • This approach speeds up result processing and prevents JVM memory overflow. • Only forward traversal is supported and the current connection cannot perform other operations until data processing completes or the statement closes. For details, see **Using the Streaming Read Function**.	Type: Boolean. Value range: • The streaming read functionality can be enabled only when this parameter is set to **true** and the statement.setFetchSize(Integer.MIN_VALUE) or statement.enableStreamingResults() API is used. • If this parameter is set to **false**, the streaming read functionality is disabled. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
preferQueryMode	Specifies the query mode.	Type: string. Value range: • **simple**: The simple query mode is used, which sends the entire SQL statement (including parameters) to the database for execution. In this mode, SQL statements cannot be cached. • **extended**: The extended query mode is used, which involves several steps—parse SQL statements, bind parameters, and execute statements. In this mode, only prepared statements are cached, while regular statements are not cached. • **extendedForPrepared**: The extended query mode is used for prepared statements, whereas the simple query mode is used for regular statements. In this mode, only prepared statements are cached, while regular statements are not cached. • **extendedCacheEverything**: The extended query mode is used, and both prepared statements and regular statements are cached. Default value: **extended**.
allowReadOnly	Specifies whether to enable the read-only mode. When **allowReadOnly** is set to **true**, the read-only mode takes effect after connection.setReadOnly(true) is called, and data cannot be modified. When **allowReadOnly** is set to **false**, the read-only mode does not take effect after connection.setReadOnly(true) is called, and data can still be modified.	Type: Boolean. Value range: • **true**: The read-only mode is enabled. • **false**: The read-only mode cannot be enabled. Default value: **true**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
parseCandidatesByDomain	Specifies whether to obtain multiple candidate nodes based on domain names.	Type: Boolean. Value range: • **true**: The domain name resolution functionality is enabled. When enabled, you need to configure the host information in the URL in the format of domain name plus port number. The driver obtains the IP address based on the domain name, generates a standby node set, and works with the **autoBalance** parameter for load balancing. • **false**: The domain name resolution functionality is disabled. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
primaryDomains	Specifies that the first *n* domain names configured in the URL are preferentially connected as the primary cluster. If the parameter value is greater than **0**, this functionality is enabled. It takes effect only when **parseCandidatesByDomain** is set to **true**. The domain names in the URL are divided into two groups. The first group is the primary cluster, and the second group is the standby cluster. After a primary/standby switchover occurs, the first group is the standby cluster, and the second group is the primary cluster. It applies to streaming DR and Dorado dual-cluster scenarios. Example: `jdbc:gaussdb://domain1:port1,domain2:port2,domain3:port3,domain4:port4/database?parseCandidatesByDomain=true&primaryDomains=2` **domain1** and **domain2** correspond to the primary cluster, and the IP addresses corresponding to **domain1** and **domain2** are preferentially connected. If a switchover occurs, **domain3** and **domain4** are marked as the primary cluster, and connections are preferentially established to **domain3** and **domain4**.	Type: integer. Value range: 0 to (Number of domain names configured in the URL – 1). Default value: **0** (disabled).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
priorityDomains	Specifies the first *n* domain names configured in the URL as the domain names to be preferentially connected. If the parameter value is greater than **0**, this functionality is enabled. It takes effect only when **parseCandidatesByDomain** is set to **true**. If **primaryDomains** has been configured, the value of this parameter must be less than that of **primaryDomains**. Example: jdbc:gaussdb://domain1:port1,domain2:port2,domain3:port3,domain4:port4/database?parseCandidatesByDomain=true&primaryDomains=2&priorityDomains=1 **domain1** and **domain2** correspond to the primary cluster, and the IP address corresponding to **domain1** is preferentially connected. **domain2** is the standby domain name. Its IP address is connected only when **domain1** cannot be connected. If a primary/standby switchover occurs, **domain3** and **domain4** will be marked as the new primary cluster. However, the IP address corresponding to **domain3** is preferentially connected. The IP address corresponding to **domain4** is connected only when **domain3** cannot be connected.	Type: integer. Value range: 0 to (Number of domain names configured in the URL – 1). Default value: **0** (disabled).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
refreshDomainResolveTime	Specifies the interval for refreshing the domain name resolution result. This parameter takes effect only when **parseCandidatesByDomain** is set to **true**. After this parameter takes effect, the domain name resolution refresh time in the URL is set to the specified value, and the domain name resolution result is updated periodically in the asynchronous thread. **NOTICE** The driver obtains domain name resolution results only from the DNS service in the environment where the application is located. To ensure that domain name binding changes can be detected by the driver in a timely manner, the DNS service in the environment where the application is located must ensure that the domain name binding changes take effect in a timely manner.	Type: integer. Unit: s. Value range: 1–2147483647. Default value: **10**.
refreshCNIpListTime	Obtains the cache validity period of the CNs. During connection establishment, the CN status in the database cluster is checked and is reliable within the period specified by **refreshCNIpListTime**. After the period expires, the status becomes invalid. The IP address list of available CNs is obtained again during the next connection establishment.	Type: integer. Unit: s. Value range: 1 to 9999. Default value: **10**. Suggestion: Adjust the value based on service requirements and use it together with **autoBalance**.
statementTimeout	Specifies the timeout value for executing a statement. If the statement execution time exceeds the value of this parameter, the current operation will be canceled.	Type: integer. Unit: ms. Value range: 0–2147483647. When the value is **0** or a negative number, the timeout functionality is disabled. Default value: **0**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
executeUpdateQueryable	Specifies whether to enable the executeUpdate() method to execute DQL statements. **NOTE** Since DQL statements are typically executed using the executeQuery() method, it is not advisable to enable this parameter.	Type: Boolean. Value range: • **true**: enabled. When enabled, **-1** is returned when the executeUpdate() method executes DQL statements, and a result set can be obtained. • **false**: disabled. Default value: **false**.
dbMonitor	Specifies whether to enable the JDBC connection monitoring functionality. **NOTE** • The connection monitoring functionality can monitor the following metrics: number of connections enabled/disabled by an application, number of abnormal disconnections, database access volume, client CPU usage, memory usage, network transmission rate, network latency, network jitter, and packet loss rate. • If **dbMonitor** is set to **true**, **loggerLevel** is set to **debug**, and **loggerFile** is set to **filePath**, the client connection monitoring data is recorded in the **filePath** log file. For details about the configuration example, see **Monitoring Database Connections**.	Type: Boolean. Value range: • **true**: enabled. • **false**: disabled. Default value: **false**.
enableGaussArrayAndStruct	Specifies whether to convert the query result whose projection column (or output parameter) is of the array or collection type into the GaussArray and GaussStruct objects.	Type: Boolean. Value range: • **true**: enabled. • **false**: disabled. Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value
getColumnsReturnNullScale	Specifies whether to return **null** for columns whose decimal places are not specified when the ResultSetMetaData#getColumns method is used to obtain the number of decimal places of the DECIMAL type.	Type: Boolean. Value range: • **true**: For columns of the DECIMAL type whose decimal places are not specified, the number of decimal places is returned as **null**. • **false**: For columns of the DECIMAL type whose decimal places are not specified, the number of decimal places is returned as **0**. Default value: **false**.
extendTablePrivilege	Specifies the open-source database whose driver behavior is compatible with the driver when the PgDatabaseMetaData#getTablePrivileges method is used to obtain user permissions on tables. Currently, only the PostgreSQL database driver is compatible.	Type: string. Value range: • **PG**: The driver behavior is compatible with the driver of the latest PostgreSQL database. Default value: **NULL**.
diffProcAndFunc	Specifies whether the DatabaseMetaData#getProcedures and DatabaseMetaData#getFunctions methods distinguish between stored procedures and functions.	Type: Boolean. Value range: • **true**: The DatabaseMetaData#getProcedures method returns only stored procedures, while the DatabaseMetaData#getFunctions method returns only functions. • **false**: Both DatabaseMetaData#getProcedures and DatabaseMetaData#getFunctions return stored procedures and functions. Default value: **false**.

5.3.2.3 Executing SQL Statements

Applications operate on a database by executing SQL statements. This section uses examples to demonstrate operations such as CREATE TABLE, INSERT, UPDATE, DELETE, and SELECT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Executing Common SQL Statements**
- **Executing a Batch INSERT Statement**
- **Executing a Prepared UPDATE Statement**
- **Executing a DELETE Statement**
- **Executing a SELECT Statement**
- **Calling a Stored Procedure**

Executing Common SQL Statements

Prerequisites: You have connected to the database using the connection object **conn**.

Create the **customer_t1** table as follows:

Step 1 Create the statement object **stmt** by calling the createStatement method of the Connection API.

```
Statement stmt = conn.createStatement();
```

Step 2 Execute the SQL statement by calling the executeUpdate method of the Statement API.

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

Step 3 Close the statement object **stmt** by calling the close method of the Statement API.

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

----End

NOTE

- When a database receives an execution request (outside a transaction block) containing multiple statements, these statements are bundled into a transaction. VACUUM is not supported within transaction blocks. If any statement fails, the entire request will be rolled back.
- When executing multiple statements through the Statement API, use semicolons (;) to separate these statements. (Stored procedures, functions, and anonymous blocks do not support executing multiple statements.) However, when **preferQueryMode** is set to **simple**, logic parsing is not performed during statement execution, making semicolons (;) ineffective as statement separators.
- A slash (/) can serve as a terminator for creating a stored procedure, function, anonymous block, or package body. However, when **preferQueryMode** is set to **simple**, logic parsing is not performed during statement execution, making slashes (/) ineffective as terminators.
- JDBC caches SQL statements in prepareStatement, which may cause memory bloat. If the JVM memory is limited, you are advised to adjust **preparedStatementCacheSizeMiB** or **preparedStatementCacheQueries**.
- When writing SQL statements, avoid placing comments immediately adjacent to SQL keywords.

Executing a Batch INSERT Statement

Prepared statements are compiled and optimized only once, but can be reused multiple times by assigning different values. This preparation helps reduce execution time. Therefore, if you need to execute an SQL operation repeatedly, make it a prepared statement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Prerequisites: The **customer_t1** table has been created by **Executing Common SQL Statements**.

To insert data in batches by executing a prepared statement, perform the following steps:

Step 1 Create the prepared statement object **pst** by calling the prepareStatement method of the Connection API.

```
PreparedStatement pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
```

Step 2 Call the corresponding API to set parameters for each data item and call the addBatch method to add SQL statements to batch processing.

```
for (int i = 0; i < 3; i++) {
  pst.setInt(1, i);
  pst.setString(2, "data " + i);
  pst.addBatch();
}
```

Step 3 Perform batch processing by calling the executeBatch method of the PreparedStatement API.

```
pst.executeBatch();
```

Step 4 Close the prepared statement object **pst** by calling the close method of the PreparedStatement API.

```
pst.close();
```

----End

NOTE

During batch processing, adhere to the following key principles to maintain database performance:

1. Continuous execution
   - The batch processing program must run continuously without interruption.
   - Any abnormal interruption significantly degrades overall database performance.
2. Optimized transaction management
   - Disable autocommit.
     ```
     conn.setAutoCommit(false);
     ```
   - Perform commits in batches rather than row by row.
   - Determine an appropriate batch size based on data volume and system resources.

Executing a Prepared UPDATE Statement

Prerequisites: Data has been inserted into the **customer_t1** table by **Executing a Batch INSERT Statement**.

To update data by executing a prepared SQL statement, perform the following steps:

Step 1 Create the prepared statement object **pstmt** by calling the prepareStatement method of the Connection API.

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

Step 2 Set parameters by calling the setString method of the PreparedStatement API.

```
pstmt.setString(1, "new Data");
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 3 Execute the prepared statement by calling the executeUpdate method of the PreparedStatement API.

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

Step 4 Close the prepared statement object **pstmt** by calling the close method of the PreparedStatement API.

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

----End

NOTICE

After the bound parameters are set in prepareStatement, a B or U packet is constructed. This packet will be sent to the server during the next step—SQL statement execution. The size of the B or U packet must not exceed the 1,023 MB length limit. Binding excessively large amounts of data at once may result in an exception due to an oversized packet. Therefore, when setting the bound parameters in prepareStatement, it is essential to assess and control the data volume to ensure compliance with this packet length limit.

Executing a DELETE Statement

Prerequisites: You have performed operations in **Executing a Prepared UPDATE Statement**.

To execute a DELETE statement, perform the following steps:

Step 1 Create the statement object **stmtd** by calling the createStatement method of the Connection API.

```
Statement stmtd = conn.createStatement();
```

Step 2 By calling the executeUpdate method of the Statement API, execute the SQL statement to delete the record with a **c_customer_sk** value of **0** from the **customer_t1** table.

```
int rd = stmtd.executeUpdate("DELETE FROM customer_t1 WHERE c_customer_sk = 0;");
```

Step 3 Close the statement object **stmtd** by calling the close method of the Statement API.

```
stmtd.close();
```

----End

Executing a SELECT Statement

Prerequisites: You have performed operations in **Executing a DELETE Statement**.

To execute a SELECT statement, perform the following steps:

Step 1 Create the statement object **stmts** by calling the createStatement() method of the Connection API.

```
Statement stmts = conn.createStatement();
```

Step 2 Execute a query statement by calling the executeQuery() method of the Statement API to return a result set.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ResultSet rs = stmts.executeQuery("SELECT * FROM customer_t1");
```

----End

Calling a Stored Procedure

GaussDB can call stored procedures through JDBC. (Prerequisites: You have connected to the database using the connection object **conn**.)

The following example demonstrates how to create a stored procedure:

```
// Create the following stored procedure (with an out parameter) in the database:
CREATE OR REPLACE PROCEDURE testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout inout integer
)
AS
BEGIN
    psv_inout := psv_in1 + psv_in2 + psv_inout;
END;
/
```

To call the stored procedure, perform the following steps:

Step 1 Create the call statement object **cstmt** by calling the prepareCall method of Connection.

```
CallableStatement cstmt = conn.prepareCall("{? = CALL testproc(?,?,?)}");
```

Step 2 Set parameters by calling the setInt method of CallableStatement.

```
cstmt.setInt(2, 50);
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

Step 3 Register an output parameter by calling the registerOutParameter method of CallableStatement.

```
cstmt.registerOutParameter(4, Types.INTEGER);  // Register an OUT parameter of the integer type.
```

Step 4 Execute the SQL statement by calling the execute method of CallableStatement.

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

Step 5 Obtain the out parameter by calling the getInt method of CallableStatement.

```
int out = cstmt.getInt(4);
```

Step 6 Close the call statement object **cstmt** by calling the close method of CallableStatement.

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

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Certain JDBC-based applications provide a named-parameter method, allowing parameters to be set by name rather than by sequence. If a parameter has a default value, you can retain the default value instead of specifying a new value. Consequently, even if the parameter sequence changes during a stored procedure, the applications require no modification. However, GaussDB's JDBC driver currently does not support this method.
- **conn.prepareCall("{? = CALL testproc(?,?,?)}")** allows you to bind parameters when executing a stored procedure. You can either bind parameters based on the placeholder sequence and register the first parameter as the output parameter, or bind parameters based on the parameter sequence in the stored procedure and register the fourth parameter as the output parameter. In the given example, the fourth parameter is registered.
- If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.
- A stored procedure and an SQL statement must be run separately.
- Inout-type parameters in a stored procedure must be registered as output parameters.

Adding Single-Shard Execution Syntaxes to Statements

Prerequisites: You have connected to the database using the connection object **conn**. You have created a **test** table and inserted data into it.

Step 1 Call the setClientInfo(String name,String value) method of the Connection object to set the **nodeName** parameter.

```
conn.setClientInfo("nodeName","datanode1");
```

Step 2 Execute the SQL statements by using the executeQuery(String sql) and execute(String sql) methods in Statement and the executeQuery() and execute() methods in PreparedStatement.

```
PreparedStatement pstm = conn.prepareStatement("SELECT * FROM test");
pstm.execute();
pstm.executeQuery();
Statement stmt=conn.createStatement();
stmt.execute("SELECT * FROM test");
stmt.executeQuery("SELECT * FROM test");
```

Step 3 Set the parameter to an empty string to disable it.

```
conn.setClientInfo("nodeName","");
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- This functionality is adapted to the single-shard execution feature. Therefore, before using it, verify that the database supports single-shard execution.
- If the **nodeName** parameter is enabled:
  - You need to manually disable it after executing SQL statements. Otherwise, the execution of other query statements will be affected.
  - All statements of the current connection will be executed on the specified DN.
  - The cache mechanism of the PreparedStatement object is affected. Specifically, cached statements will be cleared, and subsequent statements executed for single-shard queries will not be cached until the parameter is disabled to restore caching.
- The system prohibits multiple threads from modifying the **nodeName** parameter of the same connection.

5.3.2.4 Processing the Result Set

After executing SQL statements, you need to process the result set. This section is organized into the following parts: specifying a result set type, positioning a cursor in a result set, obtaining the cursor position in a result set, obtaining data from a result set, and an example procedure.

Specifying a Result Set Type

Each result set type is suited to specific scenarios. Select an appropriate result set type based on your application's needs. Before executing an SQL statement, you must create a statement object. Certain statement creation methods support specifying a result set type. The java.sql.Connection API provides the following three methods for creating a statement object and specifying a result set type:

```
// Create a Statement object. This object will generate a ResultSet object with the given type and concurrency.
createStatement(int resultSetType, int resultSetConcurrency);

// Create a PreparedStatement object. This object will generate a ResultSet object with the given type and concurrency.
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);

// Create a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency.
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-23 Result set types

Parameter	Description
resultSetType	Specifies the type of a result set. The options are as follows: • **ResultSet.TYPE_FORWARD_ONLY** (default): The ResultSet object can only be navigated forward. • **ResultSet.TYPE_SCROLL_SENSITIVE**: Modifications are visible when you scroll back to a modified row. • **ResultSet.TYPE_SCROLL_INSENSITIVE**: Modifications are not visible when you scroll back to a modified row. **NOTE** After the ResultSet object has obtained data from the database, it is insensitive to data changes made by other transactions, even if its type is **ResultSet.TYPE_SCROLL_SENSITIVE**. To obtain up-to-date data of the record pointed to by the cursor, call the refreshRow() method of the ResultSet object.
resultSetConcurrency	Specifies the concurrency of the result set. The options are as follows: • **ResultSet.CONCUR_READ_ONLY** (default): Data cannot be modified directly through the result set; the result set is read-only. • **ResultSet.CONCUR_UPDATEABLE**: The result set is updateable. Typically, you can make appropriate changes to scrollable result sets.

Positioning a Cursor in a Result Set

ResultSet objects include a cursor pointing to the current data row. The cursor is initially positioned before the first row. The next() method moves the cursor to the next row from its current position. When a ResultSet object does not have a next row, a call to this method returns **false**. Therefore, this method is used in the while loop to iterate over the result set. For scrollable result sets, the JDBC driver allows the cursor to be positioned to a specified row. **Table 5-24** describes positioning methods.

Table 5-24 Methods for positioning a cursor in a result set

Method	Description
next()	Moves the cursor to the next row from its current position in the result set.
previous()	Moves the cursor to the previous row from its current position in the result set.
beforeFirst()	Places the cursor before the first row in the result set.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
afterLast()	Places the cursor after the last row in the result set.
first()	Places the cursor to the first row in the result set.
last()	Places the cursor to the last row in the result set.
absolute(int row)	Places the cursor to a specified row in the result set.
relative(int rows)	Moves the cursor forward by the specified number of rows in the result set if **rows** is positive, or moves it backward by that number of rows if **rows** is negative.

Obtaining the Cursor Position in a Result Set

For scrollable result sets, you can call a positioning method to change the cursor position. The JDBC driver provides methods (see **Table 5-25**) for obtaining the cursor position in a result set.

Table 5-25 Methods for obtaining the cursor position in a result set

Method	Description
isFirst()	Checks whether the cursor is in the first row of the result set.
isLast()	Checks whether the cursor is in the last row of the result set.
isBeforeFirst()	Checks whether the cursor is before the first row in the result set.
isAfterLast()	Checks whether the cursor is after the last row in the result set.
getRow()	Retrieves the current row number of the cursor in the result set.

Obtaining Data from a Result Set

ResultSet objects provide a variety of methods for obtaining data from a result set. **Table 5-26** describes the most common ones. To explore other methods, refer to the JDK documentation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-26 Common methods for obtaining data from a result set

Method	Description
getInt(int columnIndex)	Retrieves int data by column index.
getInt(String columnLabel)	Retrieves int data by column name.
getString(int columnIndex)	Retrieves string data by column index.
getString(String columnLabel)	Retrieves string data by column name.
getDate(int columnIndex)	Retrieves date data by column index.
getDate(String columnLabel)	Retrieves date data by column name.

Example Procedure

After **Executing a SELECT Statement**, handle the returned result set as follows:

Step 1 Use the while loop and next() method to traverse the result set row by row.

```
while (rs.next()) {
    // Obtain values by column index.
    int c_customer_sk = rs.getInt(1);       // Obtain int data in the first column.
    String c_customer_name = rs.getString(2);      // Obtain string data in the second column.
    System.out.println("c_customer_sk: " + c_customer_sk + ", c_customer_name: " + c_customer_name);
}
```

Step 2 Call the close() method to close resources.

```
rs.close();    // Close the ResultSet object.
stmts.close();    // Close the Statement object.
```

----End

5.3.2.5 Closing the Database Connection

After completing database operations, you will need to close the database connection by calling the close() method.

```
conn.close();
```

NOTE

The Connection, Statement, and ResultSet APIs all provide a close() method, which should be used to promptly close objects after use. Closing a Connection object will indirectly close all associated Statement objects, and closing a Statement object will indirectly close its ResultSet object.

5.3.3 Typical Application Development Examples

5.3.3.1 Configuring Connection Parameters in Different Scenarios

NOTE

In the example, **host:port** represents a node, where **host** is the name or IP address of the server where the database resides, and **port** is the port number of that server.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DR

A customer has two database clusters: production cluster A and standby cluster B. After the customer performs a DR switchover, cluster A is demoted to the standby cluster, whereas cluster B is promoted to the production cluster. In this case, to avoid an application restart or re-release caused by configuration file modifications, the customer can specify clusters A and B in the connection string when initializing the configuration file. If the primary cluster is unreachable, the driver will attempt to connect to the standby cluster. Assume that cluster A has {node1, node2, node3} and cluster B has {node4, node5, node6}.

Take the dual-cluster OceanStor Dorado as an example. To connect to the new primary cluster only after the primary/standby switchover, you can set the URL as follows:

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

Note that in the streaming replication-based remote DR scenario, if the DR cluster is promoted to primary through a failover, there will be two primary clusters running at the same time. In this case, applications may fail to connect to the right new primary cluster. In such a case, you can use either of the following solutions:

- Use the preceding unified URL configuration and isolate the original primary cluster after a failover.
- Split the URL into the following two configurations based on the primary and standby clusters. This enables the service end to proactively switch the connection in the event of a failover.

  ```
  jdbc:gaussdb://node1,node2,node3/database
  jdbc:gaussdb://node4,node5,node6/database
  ```

In streaming replication–based remote DR scenarios, the DR read solution is recommended.

Refer to the following command to configure the URL of the primary cluster:

```
jdbc:gaussdb://node1,node2,node3/database?autoBalance=true&disasterToleranceCluster=true
```

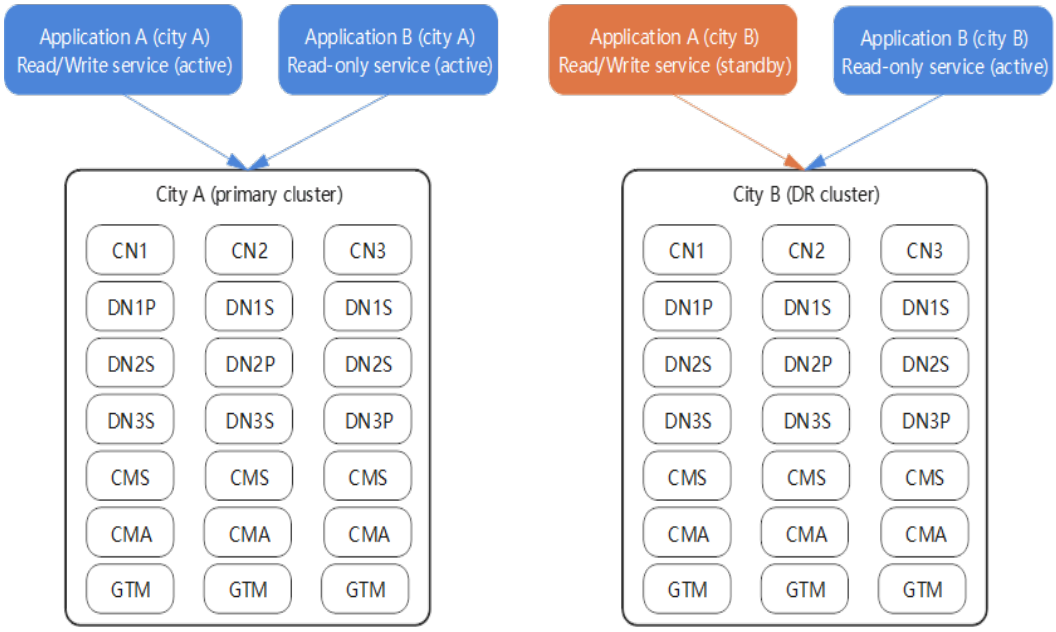
Refer to the following command to configure the URL of the DR cluster:

```
jdbc:gaussdb://node4,node5,node6/database?autoBalance=true&disasterToleranceCluster=true
```

Read/Write services and read-only services are configured for both the primary and DR clusters. Read/Write application A (city A), read-only application B (city A), and read-only application B (city B) are properly enabled, whereas service traffic for read/write application A (city B) is stopped, as shown in **Figure 5-8**.

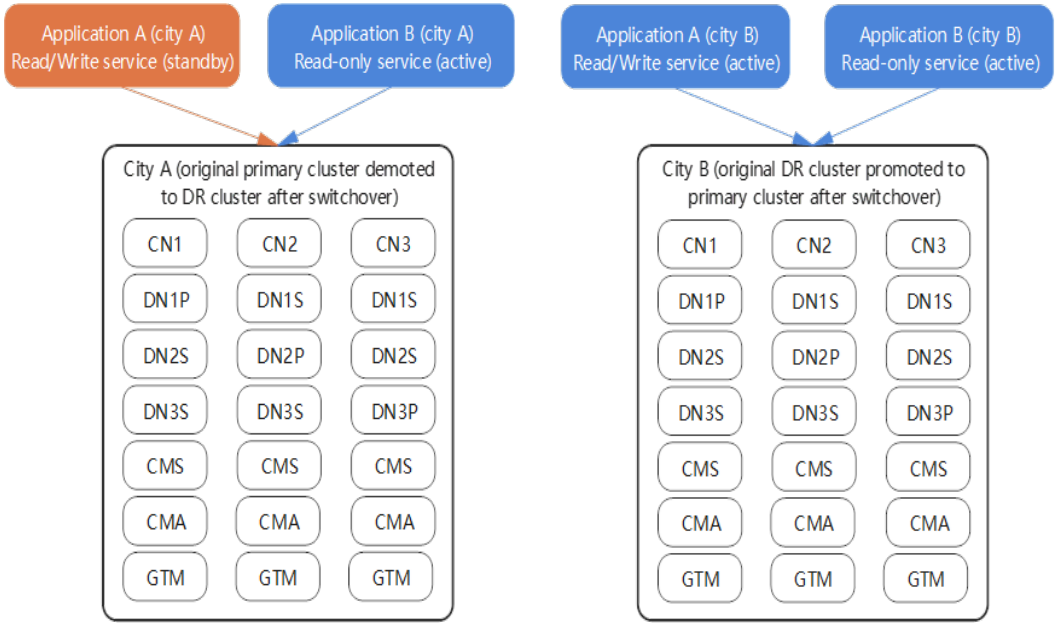
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-8 Normal operating status



In the event of a switchover, databases and applications are switched separately. In such cases, service traffic for read/write application A (city A) must be stopped first. After the switchover is completed successfully, read/write application A (city B), read-only application B (city B), and read-only application B (city A) will be properly enabled, as shown in **Figure 5-9**.

Figure 5-9 Inter-cluster switchover scenario

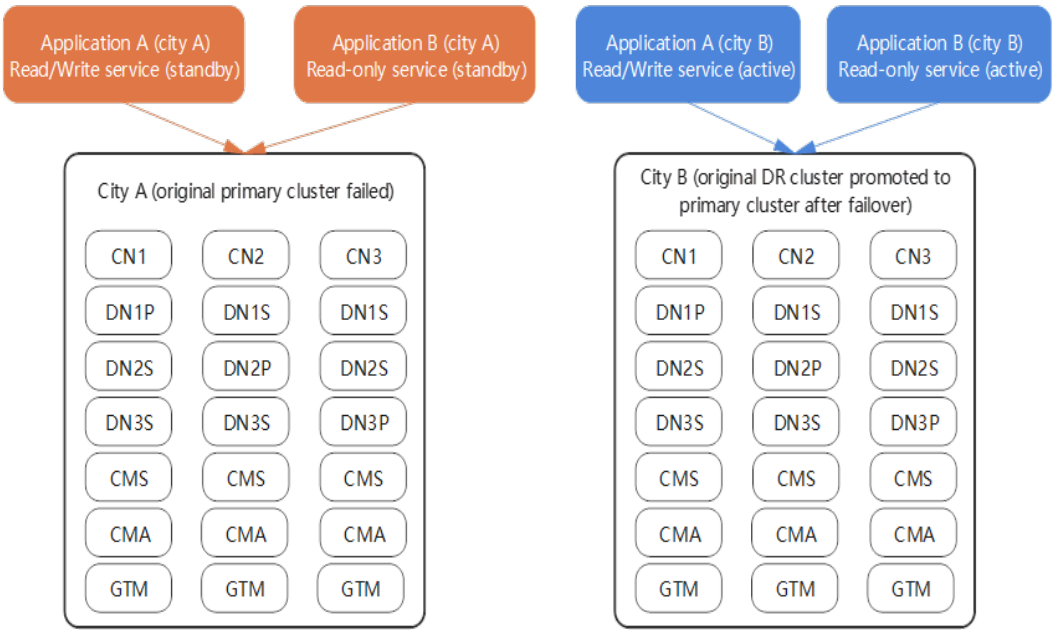


In a failover scenario where the cluster in city A becomes unavailable, service traffic for read/write application A (city A) and read-only application B (city A) must be stopped first to prevent data inconsistency. A failover is then performed to promote the cluster in city B to primary. After the cluster in city B becomes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

normal, read/write application A (city B) and read-only application B (city B) are properly enabled, whereas service traffic for read/write application A (city A) and read-only application B (city A) is stopped, as shown in **Figure 5-10**.

Figure 5-10 Failover scenario



Load Balancing

A customer has a database cluster with the following nodes: {node1, node2, node3, node4, node5, node6, node7, node8, node9, node10, node11, node12}.

- In non-multi-primary mode, the customer establishes 120 long-lived connections for Application A and expects these connections to be evenly distributed across all nodes in the cluster. In this case, refer to the following command to configure the URL:
  jdbc:gaussdb://*node1,node2,node3*/*database*?autoBalance=true
- In non-multi-primary mode, the customer develops two additional applications (B and C) and wants the three applications evenly distributed across the specified nodes. For example, Application A's connections are distributed on {node1, node2, node3, node4}, Application B's connections are distributed on {node5, node6, node7, node8}, and Application C's connections are distributed on {node9, node10, node11, node 12}. In this case, refer to the following URL configurations (if the first four nodes in the URL are unreachable, applications will connect to the fifth node, and polling will occur across all available nodes in the cluster based on information in the pgxc_node system catalog):

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

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

  Application C:
  jdbc:gaussdb://*node9,node10,node11,node12,node13*/*database*?autoBalance=priority4
- In non-multi-primary mode, the customer develops more applications using the same connection string and expects all application connections to be

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

evenly distributed across all nodes in the cluster. In this case, refer to the following command to configure the URL:
```
jdbc:gaussdb://node1,node2,node3,node4/database?autoBalance=shuffle
```

- In non-multi-primary mode, the customer develops applications that require connections to be established in single-node session mode. In this case, refer to the following command to configure the URL:
```
jdbc:gaussdb://node1,node2,node3,node4/database?
sessionType=singleNode&execTargetNode=dn_name
```
- In non-multi-primary mode, the customer chooses not to use the load balancing feature. In this case, refer to the following commands to configure the URL:
```
jdbc:gaussdb://node1/database
```

  Or
```
jdbc:gaussdb://node1/database?autoBalance=false
```

  NOTE

  When **autoBalance** is enabled in non-multi-primary mode, JDBC refreshes the list of available CNs every 10s. This default interval can be adjusted via the **refreshCNIpListTime** parameter, as shown in the example below:
```
jdbc:gaussdb://node1,node2,node3,node4/database?autoBalance=true&refreshCNIpListTime=3
```

Log Diagnosis

If customers experience slow data import or hard-to-diagnose errors, they can enable the trace log feature to facilitate diagnosis. In this case, refer to the following commands to configure the URL:

In non-multi-primary mode:
```
jdbc:gaussdb://node1/database?loggerLevel=trace
```

High Performance

Customers may execute the same SQL statement multiple times with different input parameters. To enhance execution efficiency, they can set **prepareThreshold** to avoid repeatedly generating execution plans. In this case, refer to the following commands to configure the URL:

In non-multi-primary mode:
```
jdbc:gaussdb://node1/database?prepareThreshold=5
```

When customers query 10 million rows of data at a time, returning that many rows simultaneously can cause memory overflow. To prevent this, they can set the **defaultRowFetchSize** parameter. In this case, refer to the following commands to configure the URL:

In non-multi-primary mode:
```
jdbc:gaussdb://node1/database?defaultRowFetchSize=50000
```

When inserting 10 million rows of data in batches, customers can use **batchMode** to enhance efficiency. In this case, refer to the following commands to configure the URL:

In non-multi-primary mode:
```
jdbc:gaussdb://node1/database?batchMode=on
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Case Conversion

GaussDB stores metadata in lowercase by default. When migrating metadata from a database that stores metadata in uppercase, GaussDB converts uppercase metadata to lowercase. If the original service involves the processing of uppercase metadata, you can enable **uppercaseAttributeName**. However, you are advised to modify the service code instead of using this method to solve the problem. If you have to use this function, ensure that the metadata in the current database is in lowercase to avoid problems. In this case, refer to the following commands to configure the URL:

In non-multi-primary mode:

```
jdbc:gaussdb://node1/database?uppercaseAttributeName=true
```

The APIs involved in DatabaseMetaData can be directly called based on input parameters. The methods of calling the APIs involved in ResultSetMetaData are as follows:

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM test_supper");
ResultSetMetaData rsmd = rs.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.println(rsmd.getColumnLabel(i) + "    " + rsmd.getColumnName(i));
}
```

5.3.3.2 Handling SQL Exceptions

An SQL exception will be thrown when an error occurs during JDBC operations (such as executing SQL statements or processing result sets). SQL exceptions can be handled via common methods such as getMessage() and printStackTrace(). For details, see **java.sql.SQLException**.

The following code shows how to capture SQL exceptions:

```
try {
   <some code>
   }
 } catch(SQLException e) {
     // System.out.println("exception: " + e.getMessage());      // Return the description of the error.
     e.printStackTrace ();      // Print the stack information of the current exception to the standard error
stream.
}
```

The following shows the error information displayed when an SQL exception occurs due to an incorrect column index, such as mistakenly writing rs.getString(2) as rs.getString(3) in the **Example Procedure** for processing the result set.

- Example of using the getMessage() method

  ```
  try {
     while (rs.next()) {
        int c_customer_sk = rs.getInt(1);
        String c_customer_name = rs.getString(3);      // Use an incorrect column index.
        System.out.println("c_customer_sk: " + c_customer_sk + ", c_customer_name: " +
  c_customer_name);
     }
   } catch(SQLException e) {
      System.out.println("exception: " + e.getMessage());      // Return the description of the error.
  }
  ```

  Running this program will trigger the following error information:

  ```
  exception: The column index is out of range: 3, number of columns: 2.
  ```

- Example of using the printStackTrace() method

  ```
  try {
     while (rs.next()) {
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      int c_customer_sk = rs.getInt(1);
      String c_customer_name = rs.getString(3);      // Use an incorrect column index.
      System.out.println("c_customer_sk: " + c_customer_sk + ", c_customer_name: " +
c_customer_name);
    }
 } catch(SQLException e) {
     e.printStackTrace ();      // Print the stack information of the current exception to the standard
error stream.
}
```

Running this program will trigger the following error information:

```
com.huawei.gaussdb.jdbc.util.PSQLException: The column index is out of range: 3, number of columns:
2.
    at com.huawei.gaussdb.jdbc.jdbc.PgResultSet.checkColumnIndex(PgResultSet.java:4005)
    at com.huawei.gaussdb.jdbc.jdbc.PgResultSet.checkResultSet(PgResultSet.java:4041)
    at com.huawei.gaussdb.jdbc.jdbc.PgResultSet.getString(PgResultSet.java:2754)
    at Customer.main(Customer.java:54)
```

5.3.3.3 Creating and Calling a Stored Procedure

It also demonstrates how to connect to a database and create and call stored procedures.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant
security risks. To improve security, it is advisable to store them in configuration files or environment
variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running
this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in
your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

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

public class DBTest {

 // Establish a database connection in non-encrypted mode.
 public static Connection GetConnection(String username, String passwd) {
   String driver = "com.huawei.gaussdb.jdbc.Driver";
   String sourceURL = "jdbc:gaussdb://$ip:$port/database";
   Connection conn = null;
   try {
     // Load the database driver.
     Class.forName(driver);
   } catch (Exception e) {
     e.printStackTrace();
     return null;
   }

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

   return conn;
 }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Create a stored procedure.
public static void CreateCallable(Connection conn) {
  Statement stmt = null;
  try {
    stmt = conn.createStatement();
    // Create a stored procedure and return the sum of the three input values.
    stmt.execute("CREATE OR REPLACE PROCEDURE TESTPROC \n" +
        "(\n" +
        "    psv_in1 in integer,\n" +
        "    psv_in2 in integer,\n" +
        "    psv_inout inout integer\n" +
        ")\n" +
        "as\n" +
        "begin\n" +
        "    psv_inout := psv_in1 + psv_in2 + psv_inout;\n" +
        "end;\n" +
        "/");
  } catch (SQLException e) {
    throw new RuntimeException(e);
  } finally {
    if (stmt != null) {
      try {
        stmt.close();
      } catch (SQLException e) {
        throw new RuntimeException(e);
      }
    }
  }
}

// Call the stored procedure.
public static void ExecCallableSQL(Connection conn) {
  CallableStatement cstmt = null;
  try {
    cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
    cstmt.setInt(2, 50);
    cstmt.setInt(1, 20);
    cstmt.setInt(3, 90);
    cstmt.registerOutParameter(4, Types.INTEGER);  // Register an OUT parameter of the integer type.
    cstmt.execute();
    int out = cstmt.getInt(4);  // Obtain the OUT parameter.
    System.out.println("The CallableStatement TESTPROC returns:"+out);
    cstmt.close();
  } catch (SQLException e) {
    if (cstmt != null) {
      try {
        cstmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}


/**
 * Main program. Call static methods step by step.
 * @param args
*/
public static void main(String[] args) {
  // Create a database connection.
  String userName = System.getenv("EXAMPLE_USERNAME_ENV");
  String password = System.getenv("EXAMPLE_PASSWORD_ENV");
  Connection conn = GetConnection(userName, password);

  // Create a stored procedure.
  CreateCallable(conn);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    // Call the stored procedure.
    ExecCallableSQL(conn);

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

  }

}
```

Below are the results of the preceding two examples:

```
Connection succeed!
The CallableStatement TESTPROC returns:160
```

5.3.3.4 Obtaining the Return Value of a Function

JDBC obtains the return value when calling a function. The following example shows the return values of the bit and float8 types. For other data types, refer to this example.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

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

public class Type {
    public static void main(String[] args) throws SQLException {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:gaussdb://$ip:$port/database";
        Connection conn = null;

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

        try {
            // Establish a database connection in non-encrypted mode.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Create a table.
        String createsql = "CREATE TABLE IF NOT EXISTS t_bit(col_bit bit)";
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        Statement stmt = conn.createStatement();
        stmt.execute(createsql);
        stmt.close();
        // Example of using the bit type. Note that the value range of the bit type is [0,1].
        Statement st = conn.createStatement();
        String sqlstr = "CREATE OR REPLACE FUNCTION fun_1()\n" + "RETURNS bit AS $$\n"
            + "SELECT col_bit FROM t_bit LIMIT 1;\n" + "$$\n" + "LANGUAGE SQL;";
        st.execute(sqlstr);
        CallableStatement c = conn.prepareCall("{ ? = CALL fun_1() }");
        // Register the output type, which is a bit string.
        c.registerOutParameter(1, Types.BIT);
        c.execute();
        // Use the Boolean type to obtain the result.
        System.out.println(c.getBoolean(1));

        // Example of using the float8 type.
        st.execute("CREATE TABLE IF NOT EXISTS t_float(col1 float8)");
        PreparedStatement pstm = conn.prepareStatement("INSERT INTO t_float VALUES(?)");
        pstm.setDouble(1, 123456.123);
        pstm.execute();
        pstm.close();

        // Example of using a function that returns a float8 value.
        st.execute(
            "CREATE OR REPLACE FUNCTION func_float() " + "RETURN float8 " + "AS DECLARE " + "var1
float8; " + "BEGIN "
                + " SELECT col1 INTO var1 FROM t_float LIMIT 1; " + " RETURN var1; " + "END;");
        CallableStatement cs = conn.prepareCall("{? = CALL func_float()}");
        cs.registerOutParameter(1, Types.DOUBLE);
        cs.execute();
        System.out.println(cs.getDouble(1));
        st.close();

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

Below are the results of the preceding two examples:

```
Connection succeed!
false
123456.123
```

5.3.3.5 Batch Query

In this example, setFetchSize adjusts the memory usage of the client by using the database cursor to obtain server data in batches. It may increase network interaction and damage some performance. The cursor is valid within a transaction. Therefore, disable automatic transaction commit and then manually commit the transaction.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import java.sql.Connection;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

public class Batch {
    public static void main(String[] args) throws SQLException {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:gaussdb://$ip:$port/database";
        Connection conn = null;

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

        try {
            // Establish a database connection in non-encrypted mode.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Disable automatic commit.
        conn.setAutoCommit(false);

        // Create a table.
        Statement st = conn.createStatement();
        st.execute("CREATE TABLE mytable (cal1 int);");

// Insert 200 rows of data into the table.
        PreparedStatement pstm = conn.prepareStatement("INSERT INTO mytable VALUES (?)");
        for (int i = 0; i < 200; i++) {
            pstm.setInt(1, i + 1);
            pstm.addBatch();
        }
        pstm.executeBatch();
        conn.commit();
        pstm.close();

        // Open the cursor and obtain 50 rows of data each time.
        st.setFetchSize(50);
        int fetchCount = 0;
        ResultSet rs = st.executeQuery("SELECT * FROM mytable");
        while (rs.next()) {
            fetchCount++;
        }
        System.out.println(fetchCount == 200);
        conn.commit();
        rs.close();

        // Disable the server cursor.
        st.setFetchSize(0);
        fetchCount = 0;
        rs = st.executeQuery("SELECT * FROM mytable");
        while (rs.next()) {
            fetchCount++;
        }
        System.out.println(fetchCount == 200);
        conn.commit();
        rs.close();

        // Close the statement object and database connection.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        st.close();
        conn.close();

    }
}
```

Below are the results of the preceding two examples:

```
Connection succeed!
true
true
```

Enable automatic commit.

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

5.3.3.6 Using the Streaming Read Function

During a streaming read, the server fetches all data at once and sends the data to the socket buffer of the requesting client. It pauses when the buffer is full and resumes once space becomes available. Meanwhile, the JVM reads data from the buffer one row at a time.

This approach speeds up result processing and prevents JVM memory overflow. However, streaming reads support only forward traversal; therefore, the current connection cannot perform other operations until data processing completes or the statement closes.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

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

public class Stream {
    // Create a database connection.
    public static Connection getConnection(String username, String passwd) {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableStreamingQuery=true";
        Connection conn = null;
        try {
            // Load the driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        try {
            // Connect to the database in non-encrypted mode.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return conn;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    }

    // Execute basic SQL statements to create table t_user.
    public static void CreateTable(Connection conn) {
        Statement stmt = null;
        try {
            stmt = conn.createStatement();

            // Execute basic SQL statements.
            stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
            stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
            stmt.close();
        } catch (SQLException e) {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }

    // Execute a prepared statement to insert data in batches.
    public static void BatchInsertData(Connection conn) {
        PreparedStatement pst = null;

        try {
        // Generate a prepared statement.
            pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
            for (int i = 0; i < 20; i++) {
                // Add parameters.
                pst.setInt(1, i + 1);
                pst.setString(2, "name " + (i + 1));
                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();
        }
    }

    // Enable streaming read and query the content in the t_user table.
    public static void StreamingQuery(Connection conn) {
        PreparedStatement pst = null;
        ResultSet resultSet = null;

        try {
            // Query all values in the t_user table.
            pst = conn.prepareStatement("SELECT * FROM t_user ORDER BY id",
ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
            pst.setFetchSize(Integer.MIN_VALUE);
            resultSet = pst.executeQuery();
            while (resultSet.next()) {
                System.out.println(resultSet.getInt(1));
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            if (resultSet != null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    throw new RuntimeException(e);
                }
            }

            if (pst != null) {
                try {
                    pst.close();
                } catch (SQLException e) {
                    throw new RuntimeException(e);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");

        Connection conn = getConnection(userName, password);

        CreateTable(conn);

        BatchInsertData(conn);

        StreamingQuery(conn);

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

Below are the results of the preceding two examples:

```
Connection succeed!
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- When using the streaming read feature, you must call resultSet.close() or statement.close() after processing the result set; otherwise, the current connection will become unavailable.
- During the creation of a Statement object, **resultSetType** and **resultSetConcurrency** default to **TYPE_FORWARD_ONLY** and **CONCUR_READ_ONLY**, respectively. Changing their values will render the streaming read feature ineffective.

5.3.3.7 SQL Retry at an Application Layer

If the primary DN is faulty and cannot be restored within 10s, the standby DN is automatically promoted to primary to ensure the normal running of the GaussDB cluster. Jobs running during the failover will fail and those started after the failover will not be affected. To prevent upper-layer services from being affected by the DN failover, refer to the following example to construct an SQL retry mechanism at the service layer.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

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

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }
    public void run() {
        System.out.println("exit handle");
        try {
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}

public class SQLRetry {
  // Establish a database connection in non-encrypted mode.
  public static Connection GetConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    Connection conn = null;
    try {
      // Load the database driver.
```

```
      Class.forName(driver).newInstance();
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

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

    return conn;
}

  // Execute common SQL statements to create the jdbc_test1 table.
  public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();


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

      // Execute common SQL statements.
      int rc2 = stmt
        .executeUpdate("DROP TABLE if EXISTS jdbc_test1;");

      int rc1 = stmt
        .executeUpdate("CREATE TABLE jdbc_test1(col1 INTEGER, col2 VARCHAR(10));");

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

  // Execute a prepared statement to insert data in batches.
  public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
      // Generate a prepared statement.
      pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");
      for (int i = 0; i < 100; i++) {
       // Add parameters.
        pst.setInt(1, i);
        pst.setString(2, "data " + i);
        pst.addBatch();
      }
      // Perform batch processing.
      pst.executeBatch();
      pst.close();
    } catch (SQLException e) {
      if (pst != null) {
        try {
          pst.close();
        } catch (SQLException e1) {
        e1.printStackTrace();
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      }
    }
    e.printStackTrace();
  }
}

// Run a prepared statement to query 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("finish.....");
  return retValue;
}

// Configure the number of retry attempts for the retry of a query statement upon a failure.
public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
      int maxRetryTime = 10;
      int time = 0;
      String result = null;
      do {
          time++;
          try {
System.out.println("time:" + time);
boolean ret = QueryRedo(conn);
if(ret == false){
 System.out.println("retry, time:" + time);
 Thread.sleep(10000);
 QueryRedo(conn);
}
          } catch (Exception e) {
              e.printStackTrace();
          }
      } while (null == result && time < maxRetryTime);

}

/**
 * Main program. Call static methods step by step.
 * @param args
 * @throws InterruptedException
 */
public static void main(String[] args) throws InterruptedException {
  // Create a database connection.
```

```
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    // Create a table.
    CreateTable(conn);

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

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

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

  }

}
```

Below are the results of the preceding two examples:

```
Connection succeed!
time:1
col1 = 10
finish.....
time:2
col1 = 10
finish.....
time:3
col1 = 10
finish.....
time:4
col1 = 10
finish.....
time:5
col1 = 10
finish.....
time:6
col1 = 10
finish.....
time:7
col1 = 10
finish.....
time:8
col1 = 10
finish.....
time:9
col1 = 10
finish.....
time:10
col1 = 10
finish.....
exit handle
```

5.3.3.8 Importing and Exporting Data Through Local Files

When using Java for secondary development on GaussDB, you can call the CopyManager API to stream data between the database and a local file. This lets you export data to or import data from a local file in CSV or text format.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import com.huawei.gaussdb.jdbc.copy.CopyManager;
import com.huawei.gaussdb.jdbc.core.BaseConnection;

public class Copy {

   public static void main(String[] args) {
      String urls = new String("jdbc:gaussdb://$ip:$port/database"); // Define the database URL.
     String username = System.getenv("EXAMPLE_USERNAME_ENV");           // Specify the username.
     String password = System.getenv("EXAMPLE_PASSWORD_ENV");            // Specify the password.
     String tablename = new String("migration_table"); // Define table information.
     String tablename1 = new String("migration_table_1"); // Define table information.
      String driver = "com.huawei.gaussdb.jdbc.Driver";
      Connection conn = null;

      try {
         Class.forName(driver);
         conn = DriverManager.getConnection(urls, username, password); // Connect to a database in non-encrypted mode.
      } catch (ClassNotFoundException e) {
         e.printStackTrace(System.out);
      } catch (SQLException e) {
         e.printStackTrace(System.out);
      }

    // Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.
      try {
         copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
      } catch (SQLException e) {

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

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

         e.printStackTrace();
      }

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

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

         e.printStackTrace();
      }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    }

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

        FileInputStream fileInputStream = null;

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

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

        FileOutputStream fileOutputStream = null;

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

Results of the preceding two examples: Data is consistent between the local files **data.txt** and **data1.txt** on drive D, the database table **migration_table_1**, and the database table **migration_table**.

5.3.3.9 Logical Replication (Non-Multi-Primary)

This section describes the configuration options for JDBC-based logical decoding and provides examples illustrating how to implement logical replication through JDBC APIs.

NOTICE

In the current version, use decoding on directly connected DNs because the distributed strong-consistency decoding (CN decoding) feature is disabled by default. If required, contact Huawei technical support to enable the **enable-cn-decode** parameter after evaluation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Configuration Options

For logical replication, in addition to the options outlined in **Logical Decoding Options**, you can use the configuration options designed specifically for JDBC and other streaming decoding tools, as shown below:

1. Decoder thread concurrency

   The configuration option **parallel-decode-num** specifies the number of decoder threads for parallel decoding. The value is an integer from 1 to 20. **1** indicates that decoding follows the original serial logic. Other values indicate that parallel decoding is enabled. The default value is **1**. When this parameter is set to **1**, do not configure the following options: **decode-style**, **sending-batch**, and **parallel-queue-size**.

2. Decoding format

   The configuration option **decode-style** specifies the decoding format. It can be set to the char-type value **'j'**, **'t'**, or **'b'**, representing the JSON, text, or binary format, respectively. For plug-in types other than parallel_text_decoding, parallel_json_decoding, and parallel_binary_decoding, the default value of **decode-style** is **'b'**. This option is set only when parallel decoding is allowed and binary decoding is supported only in the parallel decoding scenario. For the JSON and text formats, in batch-sent decoding results: The uint32 comprising the first four bytes in each decoding statement indicates the total byte count of the statement (excluding the four bytes occupied by the uint32 itself, with **0** signifying the end of the current decoding batch); and an 8-byte uint64 indicates the LSN (corresponding to **first_lsn** in the BEGIN scenario, **end_lsn** in the COMMIT scenario, and **lsn** in other scenarios).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

The binary encoding rules are as follows:

1. The first four bytes indicate the total byte count of the decoding result that follows—up to (but excluding) either the statement-level delimiter 'P' or the end-of-batch marker 'F'. A value of **0** signals the end of the current decoding batch.
2. The next 8-byte uint64 indicates the LSN (corresponding to **first_lsn** in the BEGIN scenario, **end_lsn** in the COMMIT scenario, and **lsn** in other scenarios).
3. The next 1-byte letter may be **B**, **C**, **I**, **U**, or **D**, representing BEGIN, COMMIT, INSERT, UPDATE, or DELETE, respectively.
4. If the 1-byte letter in step 3 is **B**:
    1. The next 8-byte uint64 indicates the CSN.
    2. The subsequent 8-byte uint64 indicates **first_lsn**.
    3. (Optional) If the next 1-byte letter is **T**, the following 4-byte uint32 indicates the length of the transaction commit timestamp, and the subsequent characters of that length form the timestamp string.
    4. (Optional) If the next 1-byte letter is **N**, the following 4-byte uint32 indicates the length of the transaction username, and the subsequent characters of that length represent the username.
    5. Additional decoding statements may still follow, separated by a 1-byte letter **P** or **F**. **P** indicates that more statements remain to be decoded in the current batch, whereas **F** marks the end of the batch.
5. If the 1-byte letter in step 3 is **C**:
    1. (Optional) If the next 1-byte letter is **X**, the following 8-byte uint64 indicates the XID.
    2. (Optional) If the next 1-byte letter is **T**, the following 4-byte uint32 indicates the timestamp length, and the subsequent characters of that length form the timestamp string.
    3. When logs are sent in batches, decoding a COMMIT log does not necessarily indicate the end of the batch. Additional transaction decoding results may still follow. If the next 1-byte letter is **P**, more transactions remain to be decoded in the current batch, whereas **F** marks the end of the batch.
6. If the 1-byte letter in step 3 is **I**, **U**, or **D**:
    1. The next 2-byte uint16 indicates the length of the schema name.
    2. The system reads the schema name based on this length.
    3. The next 2-byte uint16 indicates the length of the table name.
    4. The system reads the table name based on this length.
    5. (Optional) If the next 1-byte letter is **N**, it denotes a new tuple. If the letter is **O**, it denotes an old tuple. New tuples are sent first.
        1. The next 2-byte uint16 indicates the number of columns to be decoded for the tuple, which is recorded as **attrnum**.
        2. The following procedure is repeated for *attrnum* times:
            1. The next 2-byte uint16 indicates the length of the column name.
            2. The system reads the column name based on this length.
            3. The next 4-byte uint32 indicates the OID of the current column type.
            4. The next 4-byte uint32 indicates the length of the current column value (stored in string format). **0xFFFFFFFF** denotes NULL. A value of **0xFFFFFFFE** identifies a **rowno** system column (actual length: 8 bytes). A value of **0** denotes a zero-length string.
            5. The system reads the column value based on this length. The **rowno** system column is read as 8 bytes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6. Additional decoding statements may still follow, separated by a 1-byte letter **P** or **F**. **P** indicates that more statements remain to be decoded in the current batch, whereas **F** marks the end of the batch.

3. Decoding only on the standby node

   The configuration option **standby-connection** specifies whether to limit decoding to the standby node. The value is of the Boolean type (**0** or **1**). The value **true** (or **1**) indicates that only the standby node can be connected for decoding. When the primary node is connected for decoding, an error is reported and the system exits. The value **false** (or **0**) indicates that there is no restriction. The default value is **false** (or **0**).

4. Batch sending

   The configuration option **sending-batch** specifies whether to send results in batches. The value is either **0** or **1**. **0** indicates that decoding results are sent one by one. **1** indicates that decoding results are sent in batches when they reach 1 MB. The default value is **0**. This parameter can be set only during parallel decoding. When **sending-batch** is enabled with the decoding format set to **'j'** or **'t'**, each original decoding statement will be preceded by a uint32 value indicating the length of the decoding result (excluding the bytes occupied by the uint32 itself) and a uint64 value indicating the LSN of the result.

5. Length of the parallel decoding queue

   The configuration option **parallel-queue-size** specifies the length of the queue for interactions between parallel logical decoder threads. The value ranges from 2 to 1024 and must be a power of 2. The default value is **128**. The queue length is positively proportional to memory usage during decoding.

6. Memory threshold for logical decoding

   The configuration option **max-txn-in-memory** specifies the memory threshold (in MB) for caching the intermediate decoding result of a single transaction. In parallel decoding mode, this parameter has been deprecated and no longer takes effect. In serial decoding mode, the value range is [0,100]. The default value is **0**, indicating that the memory usage is not controlled.

   The configuration option **max-reorderbuffer-in-memory** specifies the memory threshold (in GB) for caching the intermediate decoding results of all transactions. In serial decoding mode, the value range is [0,100]. The default value is **0**, indicating that the memory usage is not controlled. In parallel decoding mode, the value range is [1,50% of **max_process_memory**]. The default value is the greater of **1** or **max_process_memory/1048576 × 10%**, where **1048576** converts the unit from KB to GB. Note that this parameter has been deprecated and no longer takes effect.

   In parallel decoding mode, the configuration option **max-logical-decode-memory** specifies the memory threshold (in GB) for caching the intermediate decoding results of all transactions. The value range is [6,1024], with a default value of **6**.

   When the memory usage exceeds the threshold, intermediate decoding results are written into a temporary file during decoding, affecting the logical decoding performance. The size of the temporary file is proportional to the amount of data modified by the transaction. If there are too many temporary files, the disk may enter the read-only state.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7. Memory threshold for the logical decoding table metadata

   The configuration option **desc-memory-limit** specifies the maximum memory overhead (in MB) for all table metadata maintained by logical decoding tasks. The value range is [10,1048576], with a default value of **200**. During decoding, if the total memory occupied by the maintained table metadata exceeds the threshold, the FIFO memory elimination mechanism of the table metadata is triggered to clear some memory.

8. Logical decoding sending timeout threshold

   The configuration option **sender-timeout** specifies the heartbeat timeout threshold between the kernel and clients. If no message is received from a client within this period, logical decoding tasks will proactively stop and disconnect from the client. The unit is ms, and the value range is [0,2147483647]. The default value depends on the value of the GUC parameter **logical_sender_timeout**. If set to **0**, logical decoding tasks will not proactively disconnect from clients. If set to a small value (for example, 1 ms), decoding tasks may be interrupted prematurely.

9. User blacklist options for logical decoding

   Use the user blacklist for logical decoding. The transaction operations of blacklisted users are filtered from the logical decoding result. The options are as follows:

   a. **exclude-userids**: specifies the OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not verify the existence of the specified OIDs. The same service user may have different OIDs on different DNs. Therefore, the OID of the service user on each DN needs to be specified for logical decoding of directly connected DNs in distributed mode. Otherwise, the logical decoding results of some DNs may be filtered while those of some DNs are not filtered.

   b. **exclude-users**: specifies blacklisted usernames. Multiple usernames are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.

   c. **dynamic-resolution**: specifies whether to dynamically parse blacklisted usernames. The default value is **true**. If this parameter is set to **false**, the system reports an error and terminates logical decoding when finding that the relevant user is not present in the blacklist (**exclude-users**). If this parameter is set to **true**, the system proceeds with decoding when finding that the relevant user is not present in the blacklist (**exclude-users**).

10. Output options for transaction logic logs

    a. **include-xids**: specifies whether the BEGIN logical log of a transaction outputs the transaction ID. The default value is **true**.

    b. **include-timestamp**: specifies whether the BEGIN logical log of a transaction outputs the time when the transaction is committed. The default value is **false**.

    c. **include-user**: specifies whether the BEGIN logical log of a transaction outputs the username of the transaction. The default value is **false**. The

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

transaction username refers to an authorized user (that is, the logged-in user executing the session associated with the transaction). This username remains unchanged throughout the execution of the transaction.

11. Logical replication slot name

    withSlotName(String slotName): uniquely identifies the logical replication slot to be created or decoded.

    NOTICE

    - Any uppercase letters in the input string will be automatically converted to lowercase.
    - The names of the logical replication slots created and used by PGReplicationStream can only consist of lowercase letters, digits, and underscores (_), with a maximum length of 63 characters.

12. Logical decoding connection options

    withStatusInterval: specifies the timeout period (default: 10,000 ms).

    NOTE

    When the primary node is under heavy load, the standby node may close its connection upon timeout. To resolve this issue, you can configure **withStatusInterval(10000,TimeUnit.MILLISECONDS)** to override **socketTimeout**.

13. Heartbeat log output option

    enable-heartbeat: specifies whether to generate heartbeat logs. The default value is **false**.

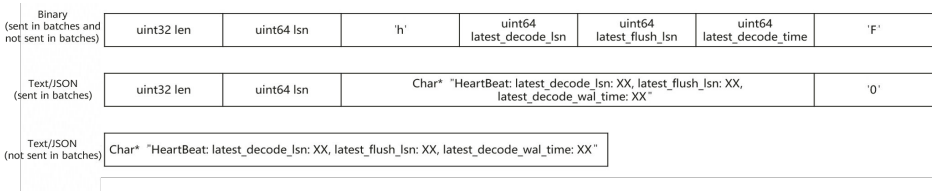
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

The following explains how heartbeat logs are parsed, with the specific format illustrated in **Figure 5-11**. (For compatibility reasons, the LSN naming convention is retained. Its actual meaning depends on the specific scenario.)

- In binary format, the log message starts with character 'h' (indicating a heartbeat log).
- It then includes the following content: an 8-byte uint64 value (during decoding on directly connected DNs, it is an LSN indicating the end position of the WAL read when the heartbeat logical log was sent; in distributed strong-consistency decoding scenarios, it represents the decoded log's transaction CSN that had been sent when the heartbeat logical log was sent); another 8-byte uint64 value (during decoding on directly connected DNs, it is an LSN indicating the position of the WAL that had been flushed to disk when the heartbeat logical log was sent; in distributed strong-consistency decoding scenarios, it represents the CSN to be assigned to the next committed transaction in the cluster); and an 8-byte uint64 timestamp (starting from January 1, 1970) recorded when the latest decoded transaction log or checkpoint log was generated.
- As for the end-of-message character, binary messages end with character 'F'. Text and JSON messages end with '0' when sent in batches and have no end-of-message character in other cases. The message content is transmitted in big-endian mode.

Figure 5-11 Heartbeat protocol format



14. (Logical decoding control parameter) Specifies the DDL reverse parsing process and output format.

    a. **enable-ddl-decoding**: The default value is **false**, indicating that logical decoding is disabled for DDL statements. **true** indicates that logical decoding is enabled for DDL statements.

    b. **enable-ddl-json-format**: The default value is **false**, indicating that DDL reverse parsing results are output in text format. **true** indicates that DDL reverse parsing results are output in JSON format.

15. Position log output option

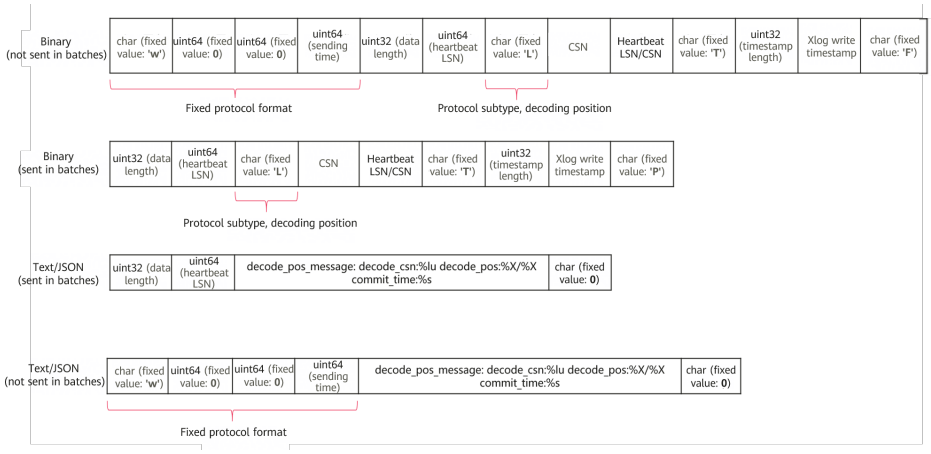
    enable-decode-position: specifies whether to output decoding position logs. The default value is **false**.

NOTE

The following explains how decoding position logs are parsed (the specific format is illustrated in **Figure 5-12**):

- In binary format, the log message starts with character 'L' (indicating a decoding position log).
- It then includes the following content: an 8-byte uint64 CSN generated when the decoding position log was sent; an 8-byte uint64 LSN (for LSN-based decoding, representing the WAL position at the time the decoding position log was sent) or CSN (for CSN-based decoding, generated when the decoding position log was sent); and an 8-byte uint64 timestamp (starting from January 1, 1970) recorded when the system function gs_add_logical_decoding_position_xlog was called.
- As for the end-of-message character, binary messages end with character 'F'. Text and JSON messages end with 0 when sent in batches and have no end-of-message character in other cases.

Figure 5-12 Heartbeat protocol format



Examples

NOTICE

- The logical replication class PGReplicationStream is not thread-safe. Multiple threads must not share the same PGReplicationStream object.
- The names of the logical replication slots created and used by PGReplicationStream can only consist of lowercase letters, digits, and underscores (_), with a maximum length of 63 characters.

Prerequisites for code execution:

1. Add the IP address of the JDBC user machine to the whitelist of the logical replication permission.
2. Set **wal_level** to **logical**. For the setting method, contact an administrator.
3. Create tables **t1** and **t2** and perform DDL or DML operations on the tables.

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in
```

```
your local environment (and adjust the variable names as needed).
// When DN decoding is applied to a table whose distribution mode is not replication, DNs decode only their local WALs. In this case, you need to connect to all DNs, create logical replication slots on each DN, and then aggregate their data. Execution on a single node will decode data only from that node, leading to incomplete log retrieval.
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.PGProperty;
import com.huawei.gaussdb.jdbc.jdbc.PgConnection;
import com.huawei.gaussdb.jdbc.replication.LogSequenceNumber;
import com.huawei.gaussdb.jdbc.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class LogicalReplicationDemo {
    private static PgConnection conn = null;
    public static void main(String[] args) {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        // Configure the database IP address and port. Here, the port refers to haPort (high-availability port), which defaults to the connected DN's port number plus 1.
        String sourceURL = "jdbc:gaussdb://$ip:$port/database";

        // Logical replication slots have a default name of replication_slot.
        // Test mode: Create a logical replication slot.
        int TEST_MODE_CREATE_SLOT = 1;
        // Test mode: Enable logical replication. (Prerequisites: A logical replication slot already exists.)
        int TEST_MODE_START_REPL = 2;
        // Test mode: Drop a logical replication slot.
        int TEST_MODE_DROP_SLOT = 3;
        // Enable different test modes.
        int testMode = TEST_MODE_START_REPL;

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

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, System.getenv("EXAMPLE_USERNAME_ENV"));
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV"));
    // For logical replication, the following three attributes are required:
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
            PGProperty.REPLICATION.set(properties, "database");
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple");
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                        .createReplicationSlot()
                        .logical()
                        .withSlotName("replication_slot") // Uppercase letters (if any) in the string are automatically converted to lowercase.
                        .withOutputPlugin("mppdb_decoding")
                        .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                // Create a replication slot before enabling this mode.
                LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("6F/E3C53568"); // LSN needs to be modified based on the actual situation.
                PGReplicationStream stream = conn
                        .getReplicationAPI()
                        .replicationStream()
                        .logical()
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                    .withSlotName("replication_slot")
                    .withSlotOption("include-xids", true)
                    .withSlotOption("skip-empty-xacts", true)
                    // .withStartPosition(waitLSN) // Specify the LSN position to start outputting decoding logs.
                    .withSlotOption("parallel-decode-num", 10) // Specify the decoder thread concurrency.
                    .withSlotOption("white-table-list", "public.t1,public.t2") // Specify the whitelist.
                   // .withSlotOption("standby-connection", true) // Force decoding on standby nodes.
                    .withSlotOption("decode-style", "t") // Specify the decoding format.
                    .withSlotOption("sending-batch", 0) // Specify whether to send decoding results in batches.
                    .withSlotOption("max-txn-in-memory", 100) // Set the memory threshold to 100 MB to
trigger flushing of a single decoded transaction.
                    .withSlotOption("max-reorderbuffer-in-memory", 2) // Set the memory threshold to 2 GB
to trigger flushing of currently decoded transactions.
                    // .withSlotOption("max-logical-decode-memory", 6) // Set the memory threshold to 6 GB
to trigger flushing of transactions undergoing parallel decoding on directly connected DNs.
                    .withSlotOption("exclude-users", "userA") // Exclude transaction logs for userA.
                    .withSlotOption("include-user", false) // Exclude usernames from the BEGIN logical logs of
transactions.
                    .withSlotOption("enable-heartbeat", true) // Enable the heartbeat log functionality.
                    .start();
            while (true) {
                ByteBuffer byteBuffer = stream.readPending();

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

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

                // If LSN flushing is required, call the following APIs as needed. Each call will trigger the
replication slot flushing logic, which may impact decoding performance. It is advisable to call these APIs
approximately every 10 seconds.
                // LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
                // stream.setFlushedLSN(lastRecv);
                // stream.forceUpdateStatus();

            }
        }else if(testMode == TEST_MODE_DROP_SLOT){
            conn.getReplicationAPI()
                    .dropReplicationSlot("replication_slot");
        }
    } catch (Exception e) {
        e.printStackTrace();
        return;
    } finally {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
// Note: The preceding code cannot directly read logical logs in binary format. These logs must be read
according to binary encoding rules.
```

Below is an example decoding result in text format (**'t'**):

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
table public t1 INSERT: a[integer]:1 b[integer]:2 c[text]:'hello'
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
table public t1 UPDATE: old-key: a[integer]:1 b[integer]:2 c[text]:'hello' new-tuple: a[integer]:1 b[integer]:5
c[text]:'hello'
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
table public t1 DELETE: a[integer]:1 b[integer]:5 c[text]:'hello'
COMMIT XID: 15506
```

Below is an example decoding result in JSON format (**'j'**):

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
{"table_name":"public.t1","op_type":"INSERT","columns_name":["a","b","c"],"columns_type":
["integer","integer","text"],"columns_val":["1","2","'hello'"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
{"table_name":"public.t1","op_type":"UPDATE","columns_name":["a","b","c"],"columns_type":
["integer","integer","text"],"columns_val":["1","5","'hello'"],"old_keys_name":["a","b","c"],"old_keys_type":
["integer","integer","text"],"old_keys_val":["1","2","'hello'"]}
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
{"table_name":"public.t1","op_type":"DELETE","columns_name":[],"columns_type":[],"columns_val":
[],"old_keys_name":["a","b","c"],"old_keys_type":["integer","integer","text"],"old_keys_val":["1","5","'hello'"]}
COMMIT XID: 15506
```

5.3.3.10 Monitoring Database Connections

The connection monitoring feature monitors these JDBC metrics: number of times that applications establish connections, number of connection closure times, number of unexpected disconnections, database access volume, client CPU usage, memory usage, uplink and downlink transmission rates, and network latency, jitter, and packet loss rate on the application-database network links. This example demonstrates how to use the connection monitoring feature of the JDBC driver.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant
security risks. To improve security, it is advisable to store them in configuration files or environment
variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running
this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in
your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

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

public class DBMonitorTest {

   public static void main(String[] args){
      String driver = "com.huawei.gaussdb.jdbc.Driver";
      String username = System.getenv("EXAMPLE_USERNAME_ENV");
      String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
      String sourceURL
         = "jdbc:gaussdb://$ip:$port/database?
dbMonitor=true&loggerLevel=debug&loggerFile=dbMonitor.log";
      try {
        // Load the database driver.
         Class.forName(driver);
      } catch (Exception e) {
         e.printStackTrace();
      }

      Connection conn = null;
      Statement statement = null;
      try {
         // Connect to the database in non-encrypted mode.
         conn = DriverManager.getConnection(sourceURL, username, passwd);
         System.out.println("Connection succeed!");
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            // Create a table.
            statement = conn.createStatement();
            String createTableQuery = "CREATE TABLE IF NOT EXISTS mytable (id INT PRIMARY KEY, name
VARCHAR(50))";
            statement.executeUpdate(createTableQuery);

            // Insert data.
            String insertQuery = "INSERT INTO mytable (id, name) VALUES (1, 'John')";
            statement.executeUpdate(insertQuery);

            // Query data.
            String selectQuery = "SELECT * FROM mytable ";
            ResultSet resultSet = statement.executeQuery(selectQuery);
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                System.out.println("id: " + id + ", name: " + name);
            }

            // Drop the table.
            String dropTableQuery = "DROP TABLE IF EXISTS mytable";
            statement.executeUpdate(dropTableQuery);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (statement != null) {
                    statement.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Below are the results of the preceding two examples:

```
Connection succeed!
id: 1, name: John
```

You can view the following connection monitoring information in the **dbMonitor.log** file:

```
Nov 23, 2023 10:30:54 AM com.huawei.gaussdb.jdbc.qos.DataProcess saveQosResult
FINE: {
  "Destination host:port" : "localhost:8000",-- IP address and port number of the server.
  "Delay" : "1.00 ms",-- Network delay.
  "Jitter" : "0.04ms",-- Network jitter.
  "Loss" : "0%",-- Network packet loss rate.
  "DownloadSpeed" : "0.395Mbps",-- Downlink rate of the network.
  "UpLoadSpeed" : "0.498Mbps"-- Uplink rate of the network.
}

Nov 23, 2023 10:30:56 AM com.huawei.gaussdb.jdbc.CollectDBData saveCollectResult
FINE: {
  "openCount": "1",-- Number of times that the application enables database connections.
  "closeCount": "1",-- Number of times that the application disables database connections.
  "abortedCount": "0",-- Number of abnormal disconnections.
  "visitCount": "12",-- Number of access requests from applications to the database.
  "cpuUsage": "20.39%",-- CPU usage of the client.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  "memoryUsage": "98.32%"-- Memory usage of the client.
}
```

5.3.3.11 Using the Array Type

This section describes how to use the array type of the JDBC driver.

Prerequisites: An ORA-compatible database is connected, and the database version is 503.0 or later.

Example 1: Using the Kernel's Tableof Type in the JDBC Driver

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.ArrayDescriptor;
import com.huawei.gaussdb.jdbc.jdbc.GaussArray;

import java.sql.*;

public class ArrayTest1 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }

    /**
     * Create a frontend database object.
     *
     * @param conn conn
     * @throws SQLException An exception occurred while executing the statement
     */
    public static void prepareTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TYPE test_array is table of int");
        stmt.execute("CREATE OR REPLACE FUNCTION test_func RETURN test_array is\n" +
                "BEGIN\n" +
                "     RETURN test_array(123, 456, 789);\n" +
                "END;");
```

```
    stmt.execute("CREATE OR REPLACE PROCEDURE test_proc(v1 in test_array, v2 out test_array) is\n" +
            "BEGIN\n" +
            "     v2 := v1;\n" +
            "     v2(3) := 456;\n" +
            "END;");
}

/**
 * Clear the database object.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing the statement
 */
public static void cleanTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("DROP PROCEDURE test_proc");
    stmt.execute("DROP FUNCTION test_func");
    stmt.execute("DROP TYPE test_array");
}

/**
 * Use array object APIs.
 *
 * @param array instance of Array
 * @throws SQLException if used array failed
 */
public static void testIntArray(Array array) throws SQLException {
    // Traverse and print an element array.
    Integer[] elements = (Integer[]) array.getArray();
    for (Integer element : elements) {
        System.out.println(element);
    }

    // Traverse and print some elements.
    Integer[] someElements = (Integer[]) array.getArray(2, 2);
    for (Integer element : someElements) {
        System.out.println(element);
    }

    // Print the element type name.
    System.out.println(array.getBaseTypeName());

    // Print whether sqlType of an element is Types.INTEGER.
    System.out.println(array.getBaseType() == Types.INTEGER);
}

/**
 * Construct an array object and use related APIs.
 *
 * @param conn conn
 * @throws SQLException if create array failed
 */
public static void testConstructArray(Connection conn) throws SQLException {
    System.out.println("===========  testConstructArray  ===========");
    String typeName = "test_array";
    ArrayDescriptor typeDesc = ArrayDescriptor.getDescriptor(typeName, conn);
    GaussArray array = new GaussArray(typeDesc, new Object[]{1, 2, 3});
    testIntArray(array);
}

/**
 * Obtain the array object returned by a function.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred
 */
public static void testReturnArrayParam(Connection conn) throws SQLException {
    System.out.println("===========  testReturnArrayParam  ===========");
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    // Run a stored procedure, obtain the array object returned by a function, and call array APIs.
     PreparedStatement stmt = conn.prepareStatement("SELECT test_func()");
     ResultSet rs = stmt.executeQuery();
     rs.next();
     Array array = rs.getArray(1);
     testIntArray(array);
  }

  /**
   * Construct an array object and input and output parameters.
   *
   * @param conn conn
   * @throws SQLException if an exception occurred
   */
  public static void testInputOutputArrayParam(Connection conn) throws SQLException {
     System.out.println("===========  testInputOutputArrayParam  ===========");
     String typeName = "test_array";
     ArrayDescriptor typeDesc = ArrayDescriptor.getDescriptor(typeName, conn);
     GaussArray inArray = new GaussArray(typeDesc, new Object[]{1, 2, 3});

     // To output parameters, enable the proc_outparam_override parameter for the database.
     Statement stmt = conn.createStatement();
     stmt.execute("SET behavior_compat_options = 'proc_outparam_override'");

     // Run a stored procedure.
     CallableStatement cstmt = conn.prepareCall("{CALL test_proc(?, ?)}");
     cstmt.setArray(1, inArray);
     cstmt.registerOutParameter(2, Types.ARRAY, typeName);
     cstmt.execute();

     // Obtain output parameters and call array APIs.
     Array outArray = cstmt.getArray(2);
     testIntArray(outArray);
  }

  /**
   * Main program. Call static methods step by step.
   *
   * @param args args
   */
  public static void main(String[] args) throws SQLException {
     // Create a database connection.
     Connection conn = getConnection();

     // Create a frontend test object.
     prepareTestObject(conn);

     // Construct an array object.
     testConstructArray(conn);

     // Return the array.
     testReturnArrayParam(conn);

     // Input and output array parameters.
     testInputOutputArrayParam(conn);

     // Delete the test object.
     cleanTestObject(conn);
  }
}
```

Below are the results of the preceding two examples:

```
===========  testConstructArray  ===========
1
2
3
2
3
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
int4
true
==========  testReturnArrayParam  ==========
123
456
789
456
789
int4
true
==========  testInputOutputArrayParam  ==========
1
2
456
2
456
int4
true
```

Example 2: Using Array Object APIs

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.ArrayDescriptor;
import com.huawei.gaussdb.jdbc.jdbc.GaussArray;

import java.sql.*;

public class ArrayTest2 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }

    /**
     * Create a frontend database object.
     *
     * @param conn conn
     * @throws SQLException An exception occurred while executing the statement
     */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    public static void prepareTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TYPE test_array is table of int");
    }

    /**
     * Clear the database object.
     *
     * @param conn conn
     * @throws SQLException if an exception occurred while executing the statement
     */
    public static void cleanTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP TYPE test_array");
    }

    /**
     * Main program and main process
     *
     * @param args args
     */
    public static void main(String[] args) throws SQLException {
        // Create a database connection.
        Connection conn = getConnection();

        // Create a frontend test object.
        prepareTestObject(conn);

        String typeName = conn.getSchema() + ".test_array";
        // Obtain the type descriptor of the test_array type.
        ArrayDescriptor desc = ArrayDescriptor.getDescriptor(typeName, conn);

        // Create a GaussArray object based on the type descriptor and element data.
        GaussArray array = new GaussArray(desc, new Object[]{1, 2, 3});

        // Obtain the array type descriptor.
        desc = array.getDescriptor();

        // Use type descriptor APIs (non-standard).
        // Print whether the type name is $currentSchema.test_array.
        System.out.println((conn.getSchema() + ".test_array").equals(desc.getSQLTypeName()));
        // Print whether sqlType of the type is Types.ARRAY.
        System.out.println(desc.getSQLType() == Types.ARRAY);

        // Use array APIs.
        // Obtain and traverse the array elements.
        Integer[] elements = (Integer[]) array.getArray();
        for (Integer element : elements) {
            System.out.println(element);
        }

        // Obtain and traverse some element arrays.
        elements = (Integer[]) array.getArray(1, 2);
        for (Integer element : elements) {
            System.out.println(element);
        }

        // Print the element type name.
        System.out.println(array.getBaseTypeName());

        // Print whether sqlType of an element is Types.INTEGER.
        System.out.println(array.getBaseType() == Types.INTEGER);

        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
true
true
1
2
3
1
2
int4
true
```

Example 3: Converting an Empty Element

When an empty element of the character type is passed to the database, it will be converted into null.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.ArrayDescriptor;
import com.huawei.gaussdb.jdbc.jdbc.GaussArray;

import java.sql.*;

public class ArrayTest3 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }

    /**
     * Create a frontend database object.
     *
     * @param conn conn
     * @throws SQLException An exception occurred while executing the statement
     */
    public static void prepareTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TYPE test_array is table of varchar(50)");
        stmt.execute("CREATE TABLE test_tab(c1 varchar(50))");
        stmt.execute("CREATE OR REPLACE PROCEDURE test_proc(v1 in test_array) is\n" +
```

```
            "BEGIN\n" +
            "     FOR i in 1..v1.count loop\n" +
            "         INSERT INTO test_tab VALUES(v1(i));\n" +
            "     END LOOP;\n" +
            "END;");
    }

    /**
     * Clear the database object.
     *
     * @param conn conn
     * @throws SQLException if an exception occurred while executing the statement
     */
    public static void cleanTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP PROCEDURE test_proc");
        stmt.execute("DROP TABLE test_tab");
        stmt.execute("DROP TYPE test_array");
    }

    /**
     * Transfer an array with empty elements to the database.
     *
     * @param conn conn
     * @throws SQLException if an exception occurred while executing the statement
     */
    public static void inParamWithEmptyElement(Connection conn) throws SQLException {
        String typeName = "test_array";
        // Obtain the type descriptor of the test_array type.
        ArrayDescriptor desc = ArrayDescriptor.getDescriptor(typeName, conn);
        // Create a GaussArray object based on the type descriptor and element data.
        GaussArray array = new GaussArray(desc, new String[]{"", "", null});
        // Traverse and print elements.
        for (String s : (String[]) array.getArray()) {
            if (s == null) {
                System.out.println("s is null");
            } else {
                if (s.isEmpty()) {
                    System.out.println("s is empty");
                }
            }
        }

        // Input parameters and run a stored procedure.
        CallableStatement cstmt = conn.prepareCall("{CALL test_proc(?)}");
        cstmt.setObject(1, array, Types.ARRAY);
        cstmt.execute();

        // Query results.
        PreparedStatement stmt = conn.prepareStatement("SELECT * FROM test_tab");
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            String s = rs.getString(1);
            if (s == null) {
                System.out.println("s is null");
            } else {
                if (s.isEmpty()) {
                    System.out.println("s is empty");
                }
            }
        }
    }

    /**
     * Main program. Call static methods step by step.
     *
     * @param args args
     */
    public static void main(String[] args) throws SQLException {
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        // Create a database connection.
        Connection conn = getConnection();

        // Create a frontend test object.
        prepareTestObject(conn);

        // Transfer an array with empty elements to the database as input parameters.
        inParamWithEmptyElement(conn);

        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

```
s is empty
s is empty
s is null
s is null
s is null
s is null
```

Example 4: Output Parameter Overloading

When the registered VARRAY type differs from the VARRAY type of a stored procedure's output parameter, the actual instance type returned in the output parameter matches the output parameter type of the stored procedure.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.GaussArray;

import java.sql.*;

public class ArrayTest4 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
}

/**
 * Create a frontend database object.
 *
 * @param conn conn
 * @throws SQLException An exception occurred while executing the statement
 */
public static void prepareTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("CREATE OR REPLACE PACKAGE test_pkg IS\n" +
            "    TYPE t1 IS varray(20) OF INT;\n" +
            "    TYPE t2 IS varray(20) OF VARCHAR(50);\n" +
            "    PROCEDURE proc_out_t1(v out t1);\n" +
            "END test_pkg;");

    stmt.execute("CREATE OR REPLACE PACKAGE BODY test_pkg IS\n" +
            "    PROCEDURE proc_out_t1(v out t1) IS\n" +
            "    BEGIN\n" +
            "        v := t1(1, 2, 3, 4, 5);\n" +
            "    END;\n" +
            "END test_pkg;");
}

/**
 * Clear the database object.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing the statement
 */
public static void cleanTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("DROP PACKAGE test_pkg");
}

/**
 * Register an overloading type.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing statement
 */
public static void testRegisterDiffType(Connection conn) throws SQLException {
    // To output parameters, enable the proc_outparam_override parameter for the database.
    Statement stmt = conn.createStatement();
    stmt.execute("SET behavior_compat_options = 'proc_outparam_override'");
    CallableStatement cstmt = conn.prepareCall("{CALL test_pkg.proc_out_t1(?)}");

    // Register the t2 type.
    cstmt.registerOutParameter(1, Types.ARRAY, "test_pkg.t2");
    cstmt.execute();

    // Obtain the t1 type with the return type of getArray being Integer[].
    GaussArray array = (GaussArray)cstmt.getArray(1);
    // Traverse elements.
    for (Integer element : (Integer[]) array.getArray()) {
        System.out.println(element);
    }
}

/**
 * Main program. Call static methods step by step.
 *
 * @param args args
 */
public static void main(String[] args) throws SQLException {
    // Create a database connection.
    Connection conn = getConnection();

    // Create a frontend test object.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        prepareTestObject(conn);

        // Register a different type.
        testRegisterDiffType(conn);

        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

```
1
2
3
4
5
```

NOTICE

- The type name is case-sensitive and cannot contain decimal points.
- The type name does not support synonyms.
- The type name can be in the *Schema*.*Package*.*Type*, *Package*.*Type*, *Schema*.*Type*, or *Type* format. If no schema name is specified, the current schema is used by default. If the schema or package name contains a decimal point, it must be enclosed in double quotation marks.
- To construct ArrayDescriptor using the built-in native array type, the input type name must contain the schema name (the current schema does not have the built-in native array type).
- The element supports the following basic types: int2, int4, int8, float4, float8, numeric, bool, bpchar, varchar, nvarchar2, name, text, timestamp, timestamptz, time, timetz, clob, bytea, and blob. Possible aliases of the preceding types include smallint, int, integer, bigint, number, float, boolean, char, and varchar2.
- The element supports the following user-defined types: array, collection, and record.
- After JDBC enables the URL parameter **enableGaussArrayAndStruct**, the output parameters and return values of the collection or array type can be obtained as array objects. To output parameters, the GUC parameter **behavior_compat_options** needs to be set to **'proc_outparam_override'** for the database.
- The output parameter or return value cannot be a multi-dimensional array. The array start index of the output parameter or return value cannot be 1.
- JDBC does not verify type modifiers and relies on the database for verification. For example:
  1. It does not verify element length. Taking the "table of varchar(5)" type as an example, when constructing an array with strings longer than five characters, JDBC does not report an error.
  2. It does not verify element precision. Taking the "table of numeric(3)" type as an example, when constructing an array with numbers exceeding three digits, JDBC does not report an error.
  3. It does not convert element precision. Taking the "varray of numeric(3, 1)" type as an example, when constructing an array with a numeric value like **2.55**, JDBC does not convert it to **2.6**.
  4. It does not verify varray length. Taking the "varray(2) of int" type as an example, when constructing an array with more than two elements, JDBC does not report an error.
- When an array object passed to the database contains empty elements, those elements will be converted into null. For example, when an empty element of the character type is passed to the database, it will be converted into null. The same applies to a CLOB element containing character strings of length 0, a BLOB element containing binary arrays of length 0, and a BYTEA element containing binary arrays of length 0. In all these cases, the database will convert the elements into null.

  You are advised not to store empty elements in the constructed array object, because empty elements will be converted into null when being passed to the database. For details, see **Example 3: Converting an Empty Element**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If an array object passed from the database to the client contains empty elements, the client also converts these empty elements into null.

- Implicit conversion is supported between different array types. When the registered type differs from the output parameter type of a stored procedure, the actual instance type returned in the output parameter conforms to the output parameter type of the stored procedure. For details, see **Example 4: Output Parameter Overloading**.
- For details about array APIs, see the description of standard APIs supported by GaussArray objects.
- ArrayDescriptor provides non-standard APIs. The getSQLType API always returns **java.sql.Types.ARRAY**.

  For the type names returned by the getSQLTypeName API of ArrayDescriptor, refer to **Data Type Mapping**.

5.3.3.12 Using the Struct Type

This section describes how to use the struct type of the JDBC driver.

Prerequisites: An ORA-compatible database is connected, and the database version is 503.0 or later.

Example 1: Using the Kernel's Record Type in the JDBC Driver

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.StructDescriptor;
import com.huawei.gaussdb.jdbc.jdbc.GaussStruct;

import java.sql.*;

public class StructTest1 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            return null;
        }

        return conn;
    }

    /**
     * Create a frontend database object.
     *
     * @param conn conn
     * @throws SQLException An exception occurred while executing the statement
     */
    public static void prepareTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        // To output parameters, enable the proc_outparam_override parameter for the database.
        stmt.execute("SET behavior_compat_options = 'proc_outparam_override'");
        // Create a test object.
        stmt.execute("CREATE OR REPLACE PACKAGE test_pkg IS\n" +
                "    TYPE test_rec IS record(col1 int, col2 varchar(50));\n" +
                "    FUNCTION test_func RETURN test_rec;\n" +
                "    PROCEDURE test_proc(v1 in test_rec, v2 out test_rec);\n" +
                "END test_pkg;");
        stmt.execute("CREATE OR REPLACE PACKAGE BODY test_pkg IS\n" +
                "    FUNCTION test_func RETURN test_rec IS\n" +
                "        v test_rec;" +
                "    BEGIN\n" +
                "        v.col1 := 123;\n" +
                "        v.col2 := 'abc';\n" +
                "        RETURN v;\n" +
                "    END;\n" +
                "    PROCEDURE test_proc(v1 in test_rec, v2 out test_rec) IS\n" +
                "    BEGIN\n" +
                "        v2.col1 := v1.col1 + 1;\n" +
                "        v2.col2 := v1.col2 || 'd';\n" +
                "    END;\n" +
                "END test_pkg;");
    }

    /**
     * Clear the database object.
     *
     * @param conn conn
     * @throws SQLException if an exception occurred while executing the statement
     */
    public static void cleanTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP PACKAGE test_pkg");
    }

    /**
     * Use struct object APIs.
     *
     * @param struct instance of Struct
     * @throws SQLException if used Struct failed
     */
    public static void testStruct(Struct struct) throws SQLException {
        // Traverse and print elements.
        Object[] attributes = struct.getAttributes();
        for (Object attribute : attributes) {
            System.out.println(attribute);
        }
    }

    /**
     * Construct a struct object and use related APIs (non-standard).
     *
     * @param conn conn
     * @throws SQLException if create Struct failed
     */
```

```
public static void testConstructStruct1(Connection conn) throws SQLException {
    System.out.println("==========  testConstructStruct1  ==========");
    String typeName = "test_pkg.test_rec";
    StructDescriptor typeDesc = StructDescriptor.getDescriptor(typeName, conn);
    GaussStruct struct = new GaussStruct(typeDesc, new Object[]{666, "aaabbbccc"});
    testStruct(struct);
}

/**
 * Construct a struct object and use related APIs (standard).
 *
 * @param conn conn
 * @throws SQLException if create Struct failed
 */
public static void testConstructStruct2(Connection conn) throws SQLException {
    System.out.println("==========  testConstructStruct2  ==========");
    String typeName = "test_pkg.test_rec";
    Struct struct = conn.createStruct(typeName, new Object[]{666, "aaabbbccc"});
    testStruct(struct);
}

/**
 * Obtain the struct object returned by a function.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred
 */
public static void testReturnStructParam(Connection conn) throws SQLException {
    System.out.println("==========  testReturnStructParam  ==========");

   // Run a stored procedure, obtain the struct object returned by a function, and call struct APIs.
    PreparedStatement stmt = conn.prepareStatement("SELECT test_pkg.test_func()");
    ResultSet rs = stmt.executeQuery();
    rs.next();
    Struct Struct = (Struct) rs.getObject(1);
    testStruct(Struct);
}

/**
 * Construct a struct object and input and output parameters.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred
 */
public static void testInputOutputStructParam(Connection conn) throws SQLException {
    System.out.println("==========  testInputOutputStructParam  ==========");
    String typeName = "test_pkg.test_rec";
    StructDescriptor typeDesc = StructDescriptor.getDescriptor(typeName, conn);
    GaussStruct inStruct = new GaussStruct(typeDesc, new Object[]{123, "abc"});

    // Run a stored procedure.
    CallableStatement cstmt = conn.prepareCall("{CALL test_pkg.test_proc(?, ?)}");
    cstmt.setObject(1, inStruct);
    cstmt.registerOutParameter(2, Types.STRUCT, typeName);
    cstmt.execute();

    // Obtain output parameters and call struct APIs.
    Struct outStruct = (Struct) cstmt.getObject(2);
    testStruct(outStruct);
}

/**
 * Main program. Call static methods step by step.
 *
 * @param args args
 */
public static void main(String[] args) throws SQLException {
    // Create a database connection.
    Connection conn = getConnection();
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        // Create a frontend test object.
        prepareTestObject(conn);

        // Method 1 for constructing a struct object
        testConstructStruct1(conn);

        // Method 2 for constructing a struct object
        testConstructStruct2(conn);

        // Return the struct.
        testReturnStructParam(conn);

        // Input and output struct parameters.
        testInputOutputStructParam(conn);

        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

```
==========  testConstructStruct1  ==========
666
aaabbbccc
==========  testConstructStruct2  ==========
666
aaabbbccc
==========  testReturnStructParam  ==========
123
abc
==========  testInputOutputStructParam  ==========
124
abcd
```

Example 2: Using Struct Object APIs

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import com.huawei.gaussdb.jdbc.jdbc.GaussStruct;
import com.huawei.gaussdb.jdbc.jdbc.StructDescriptor;

import java.sql.*;

public class StructTest2 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
```

```
    try {
        // Create a database connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }

    return conn;
}

/**
 * Create a frontend database object.
 *
 * @param conn conn
 * @throws SQLException An exception occurred while executing the statement
 */
public static void prepareTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("CREATE OR REPLACE PACKAGE test_pkg IS\n" +
            "    TYPE test_rec IS record(col1 int, col2 varchar(50));\n" +
            "END test_pkg;");
}

/**
 * Clear the database object.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing the statement
 */
public static void cleanTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("DROP PACKAGE test_pkg");
}

/**
 * Main program. Call static methods step by step.
 *
 * @param args args
 */
public static void main(String[] args) throws SQLException {
    // Create a database connection.
    Connection conn = getConnection();

    // Create a frontend test object.
    prepareTestObject(conn);

    String typeName = conn.getSchema() + ".test_pkg.test_rec";
    // Obtain the type descriptor of the test_array type.
    StructDescriptor desc = StructDescriptor.getDescriptor(typeName, conn);

    // Create a GaussStruct object based on the type descriptor and element data.
    GaussStruct struct = new GaussStruct(desc, new Object[]{123, "abc"});

    // Obtain the record type descriptor.
    desc = struct.getDescriptor();

    // Use type descriptor APIs (non-standard).
    // Print whether the type name is $currentSchema.test_pkg.test_rec.
    System.out.println(typeName.equals(desc.getSQLTypeName()));
    // Print whether sqlType of the type is Types.STRUCT.
    System.out.println(desc.getSQLType() == Types.STRUCT);

    // Use a struct API.
    // Obtain and traverse the struct elements.
    Object[] attributes = struct.getAttributes();
    for (Object attribute : attributes) {
        System.out.println(attribute);
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

```
true
true
123
abc
```

Example 3: Converting an Empty Element

When an empty element of the character type is passed to the database, it will be converted into null.

Example in non-multi-primary mode:

```
// Writing the authentication credentials—username and password—directly in the code creates significant security risks. To improve security, it is advisable to store them in configuration files or environment variables instead. (Keep the password encrypted and decrypt it only when necessary.)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
// Provide actual values for $ip, $port, and database.

import java.sql.*;

public class StructTest3 {
    // Establish a database connection in non-encrypted mode.
    public static Connection getConnection() {
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableGaussArrayAndStruct=true";
        Connection conn = null;
        try {
            // Load the database driver.
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        return conn;
    }

    /**
     * Create a frontend database object.
     *
     * @param conn conn
     * @throws SQLException An exception occurred while executing the statement
     */
    public static void prepareTestObject(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TABLE test_tab(c1 varchar(50))");

        stmt.execute("CREATE OR REPLACE PACKAGE test_pkg IS\n" +
```

```
            "    TYPE test_rec IS record(col1 varchar(50), col2 varchar(50), col3 varchar(50));\n" +
            "    PROCEDURE test_proc(v1 in test_rec);\n" +
            "END test_pkg;");

    stmt.execute("CREATE OR REPLACE PACKAGE BODY test_pkg IS\n" +
            "    PROCEDURE test_proc(v1 in test_rec) IS\n" +
            "    BEGIN\n" +
            "        INSERT INTO test_tab VALUES(v1.col1);\n" +
            "        INSERT INTO test_tab VALUES(v1.col2);\n" +
            "        INSERT INTO test_tab VALUES(v1.col3);\n" +
            "    END;\n" +
            "END test_pkg;");
}

/**
 * Clear the database object.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing the statement
 */
public static void cleanTestObject(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    stmt.execute("DROP PACKAGE test_pkg");
    stmt.execute("DROP TABLE test_tab");
}

/**
 * Transfer a struct with empty elements to the database.
 *
 * @param conn conn
 * @throws SQLException if an exception occurred while executing the statement
 */
public static void inParamWithEmptyElement(Connection conn) throws SQLException {
    String typeName = "test_pkg.test_rec";
    // Create a struct object.
    Struct struct = conn.createStruct(typeName, new Object[]{"", "", null});
    // Traverse and print elements.
    for (Object attribute : struct.getAttributes()) {
        if (attribute == null) {
            System.out.println("attribute is null");
        } else {
            if (((String) attribute).isEmpty()) {
                System.out.println("attribute is empty");
            }
        }
    }

    // Input parameters and run a stored procedure.
    CallableStatement cstmt = conn.prepareCall("{CALL test_pkg.test_proc(?)}");
    cstmt.setObject(1, struct, Types.STRUCT);
    cstmt.execute();

    // Query results.
    PreparedStatement stmt = conn.prepareStatement("SELECT * FROM test_tab");
    ResultSet rs = stmt.executeQuery();
    while (rs.next()) {
        String s = rs.getString(1);
        if (s == null) {
            System.out.println("s is null");
        } else {
            if (s.isEmpty()) {
                System.out.println("s is empty");
            }
        }
    }
}

/**
 * Main program. Call static methods step by step.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
     *
     * @param args args
     */
    public static void main(String[] args) throws SQLException {
        // Create a database connection.
        Connection conn = getConnection();

        // Create a frontend test object.
        prepareTestObject(conn);

        // Transfer a struct with empty elements to the database as input parameters.
        inParamWithEmptyElement(conn);

        // Delete the test object.
        cleanTestObject(conn);
    }
}
```

Below are the results of the preceding two examples:

```
attribute is empty
attribute is empty
attribute is null
s is null
s is null
s is null
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- The type name is case-sensitive and cannot contain decimal points.
- The type name does not support synonyms.
- The type name can be in the *Schema*.*Package*.*Type*, *Package*.*Type*, *Schema*.*Type*, or *Type* format. If no schema name is specified, the current schema is used by default. If the schema or package name contains a decimal point, it must be enclosed in double quotation marks.
- The element supports the following basic types: int2, int4, int8, float4, float8, numeric, bool, bpchar, varchar, nvarchar2, name, text, timestamp, timestamptz, time, timetz, clob, bytea, and blob. Possible aliases of the preceding types include smallint, int, integer, bigint, number, float, Boolean, char, and varchar2.
- The element supports the following user-defined types: array, collection, and record.
- After JDBC enables the URL parameter **enableGaussArrayAndStruct**, the output parameters and return values of the collection or array type can be obtained as struct objects. To output parameters, the GUC parameter **behavior_compat_options** needs to be set to **'proc_outparam_override'** for the database.
- JDBC does not verify type modifiers and relies on the database for verification. For example:
  1. It does not convert element precision. Taking the "record(c1 numeric(3, 1), c2 numeric(3, 1))" type as an example, when constructing a struct with a numeric value like **2.55**, JDBC does not convert it to **2.6**.
  2. It does not verify element length. Taking the "record(c1 varchar(5), c2 varchar(5))" type as an example, when constructing a struct with strings longer than five characters, JDBC does not report an error.
  3. It does not verify elements against the NOT NULL constraint. Taking the "record(c1 int not null, c2 int)" type as an example, when constructing a struct with a null element in the first column, JDBC does not report an error.
- When a struct object passed to the database contains empty elements, those elements will be converted into null. For example, when an empty element of the character type is passed to the database, it will be converted into null. The same applies to a CLOB element containing character strings of length 0, a BLOB element containing binary arrays of length 0, and a BYTEA element containing binary arrays of length 0. In all these cases, the database will convert the elements into null.

  You are advised not to store empty elements in the constructed struct object, because empty elements will be converted into null when being passed to the database. For example, empty elements will be converted into null. For details, see Example 3.

  If a struct object passed from the database to the client contains empty elements, the client also converts these empty elements into null.
- For details about struct APIs, see the description of standard APIs supported by GaussStruct objects.
- StructDescriptor is a non-standard API. The getSQLType API of StructDescriptor always returns **java.sql.Types.STRUCT**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about the type name returned by the getSQLTypeName API of StructDescriptor, see **Data Type Mapping**.

5.3.4 JDBC API Reference

This section describes common JDBC APIs. For more APIs, check JDK 1.8 (software package) and JDBC 4.2.

5.3.4.1 Common Packages and APIs

Table 5-27 describes the functionalities of common packages and APIs.

Table 5-27 Common packages and APIs

Common Package	Functionality	Common API
java.sql	The **java.sql** package is the core and most widely used package of JDBC. It provides standard APIs for database access. You can use these APIs to connect to databases, execute SQL statements, and process results.	• Connection: connects to databases. • DriverManager: manages the loading of the JDBC driver and its connection establishment. • Statement: executes static SQL statements and returns results. • PreparedStatement: executes prepared SQL statements. • CallableStatement: executes stored procedures. • ResultSet: processes the result sets returned from database queries. • ResultSetMetaData: describes ResultSet objects. • DatabaseMetaData: describes database-specific information. • SQLException: handles exceptions during database operations. • Array, BLOB, CLOB, and NCLOB: process large object data types in databases.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Common Package	Functionality	Common API
javax.sql	The **javax.sql** package is dedicated to delivering enterprise-grade services, providing more advanced features such as connection pool, distributed transaction, and RowSet.	• DataSource: provides a flexible way to obtain database connections and supports advanced features such as connection pooling. • ConnectionPoolData-Source: provides access to data source connection pools. • PooledConnection: creates connections in connection pools. • RowSet: defines a scrollable result set.
java.util	The **java.util** package is a core utility package in the Java standard library. It provides the APIs and classes required by common data structures, algorithms, and utilities.	• Collection: defines basic operations for all single-column collections (lists, sets, and queues). • Map: defines a collection of key-value pairs. • Properties: processes key-value pair configurations, especially those with string keys and values.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Common Package	Functionality	Common API
java.io	The **java.io** package provides all the APIs and classes required for processing input/output operations.	• InputStream: defines the top-level API for byte-based input streams. This API reads data in bytes. • OutputStream: defines the top-level API for byte-based output streams. This API writes data in bytes. • Reader: defines the top-level API for character-based input streams. This API reads data in characters. • Writer: defines the top-level API for character-based output streams. This API writes data in characters. • IOException: handles errors or interruptions during input/output operations.
com.huawei.gaussdb.jdbc	This package serves as an extension to JDBC in GaussDB.	• CopyManager: implements batch data import/export for databases. • PGReplicationConnection: implements logical replication functionalities. • PGReplicationStream: operates on logical replication streams. • ChainedStreamBuilder: builds replication streams. • ChainedCommonStream-Builder: specifies common parameters for logical and physical replication.

5.3.4.2 java.sql.Connection

java.sql.Connection is an API for connecting to a database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-28 Support for java.sql.Connection

Method Name	Description	Return Type	Throws	Support JDBC 4
abort(Executor executor)	Aborts an open connection.	Void	SQLException	Yes
clearWarnings()	Clears all warnings reported for this connection object.	Void	SQLException	Yes
close()	Releases the database and JDBC resources of this connection object immediately instead of waiting for them to be automatically released.	Void	SQLException	Yes
commit()	Makes all changes made since the last commit/rollback permanent and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	Void	SQLException	Yes
createArrayOf(String typeName, Object[] elements)	Creates an array object using this factory method.	Array	SQLException	Yes
createBlob()	Constructs an object that implements the BLOB API. The returned object initially contains no data, but you can add data to it by calling the setBinaryStream and setBytes methods of the BLOB API.	BLOB	SQLException	Yes
createClob()	Constructs an object that implements the CLOB API. The returned object initially contains no data, but you can add data to it by calling the setAsciiStream, setCharacterStream, and setString methods of the CLOB API.	CLOB	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
createSQLXML()	Constructs an object that implements the SQLXML API. The returned object initially contains no data, but you can add data to it by calling the createXmlStream-Writer object and setString method of the SQLXML API.	SQLXML	SQLException	Yes
createStatement()	Creates a statement object that sends SQL statements to a database.	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes
getAutoCommit()	Retrieves the current autocommit mode for this connection object.	Boolean	SQLException	Yes
getCatalog()	Retrieves the current catalog name of this Connection object.	String	SQLException	Yes
getClientInfo()	Returns a list containing the name and current value of each client information property supported by the driver. If the client information property has not been set and has no default value, the value of this attribute may be **NULL**. Currently, only ApplicationName and ApplicationType are obtained.	Properties	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
getClientInfo(String name)	Returns the value of the client information property specified by name. This method may return **NULL** if the specified client information property has not been set and there is no default value. This method also returns **NULL** if the driver does not support the specified client information property name.	String	SQLException	Yes
getHoldability()	Retrieves the current holdability of the ResultSet object created using this connection object.	Int	SQLException	Yes
getMetaData()	Retrieves the DatabaseMetaData object that contains metadata about the database to which this connection object represents the connection. Metadata includes information about database tables, supported SQL syntax, stored procedures, and functions of the connection.	DatabaseMetaData	SQLException	Yes
getNetworkTimeout()	Obtains the number of milliseconds the driver will wait for a database request to complete. If the limit is exceeded, SQLException is thrown.	Int	SQLException	Yes
getSchema()	Retrieves the current schema name of this Connection object.	String	SQLException	Yes
getTransactionIsolation()	Retrieves the current transaction isolation level of this Connection object.	Int	SQLException	Yes
getTypeMap()	Retrieves the map object associated with this Connection object. The returned type mapping remains empty unless applications add entries.	Map<String,Class<?>>	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
getWarnings()	Retrieves the first warning reported by calls on this Connection object. If multiple warnings occur, each subsequent warning links to the first and can be retrieved by calling the SQLWarning.getNextWarning method on the previous warning.	SQLWarning	SQLException	Yes
isClosed()	Retrieves whether this Connection object is closed. It is closed when you call its close method or when a major error occurs. This method returns **true** only when the Connection.Close method has been called first.	Boolean	SQLException	Yes
isReadOnly()	Retrieves whether this Connection object is read-only.	Boolean	SQLException	Yes
isValid(int timeout)	Returns **true** if this connection remains open and valid. When this method is called, the driver should execute a query or use other mechanisms to affirmatively verify that the connection is still valid.	Boolean	SQLException	Yes
nativeSQL(String sql)	Before sending a JDBC SQL statement, the driver may convert it according to native SQL syntax. This method returns the native SQL statement generated by the driver.	String	SQLException	Yes
prepareCall(String sql)	Creates a CallableStatement object to call a stored procedure. The CallableStatement object provides methods for setting its IN and OUT parameters and for executing stored procedure calls.	CallableStatement	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method works like the previous prepareCall method, but allows you to overwrite the default result set type and concurrency.	CallableStatement	SQLException	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method works like the previous prepareCall method, but allows you to overwrite the default result set type, concurrency, and holdability.	CallableStatement	SQLException	Yes
prepareStatement(String sql)	Creates a prepared statement object to send parameterized SQL statements to the database.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int autoGeneratedKeys)	Creates a default prepared statement object that retrieves automatically generated keys. The given constant tells the driver whether it should make the auto-generated keys available for retrieval. The driver ignores this parameter if the SQL statement is neither an INSERT nor one that can return automatically generated keys.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int[] columnIndexes)	Creates a default prepared statement object that returns automatically generated keys specified by the given array. This array lists the column indexes of the automatically generated keys that should be available. The driver ignores this array if the SQL statement is neither an INSERT nor one that can return automatically generated keys.	PreparedStatement	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Creates a prepared statement object that generates a ResultSet object with the given type and concurrency. This method works like the previous prepareStatement method, but allows you to overwrite the default result set type and concurrency.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a prepared statement object that generates a ResultSet object with the given type, concurrency, and holdability.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, String[] columnNames)	Creates a default prepared statement object that returns automatically generated keys specified by the given array. This array lists the column names of the automatically generated keys that the target table needs to return. The driver ignores this array if the SQL statement is neither an INSERT nor one that can return automatically generated keys.	PreparedStatement	SQLException	Yes
releaseSavepoint(Savepoint savepoint)	Deletes the specified savepoint and subsequent savepoint objects from the current transaction. Any reference to the savepoint after it is deleted will cause an SQL exception to be thrown.	Void	SQLException	Yes
rollback()	Undoes all changes made in the current transaction and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	Void	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
rollback(Savepoint savepoint)	Undoes all changes made after a given savepoint object is set. This method should be used only when the autocommit mode is disabled.	Void	SQLException	Yes
setAutoCommit(boolean autoCommit)	Sets the autocommit mode of this connection to the given state. If a connection is in autocommit mode, all its SQL statements are executed and committed as a single transaction. Otherwise, its SQL statements are grouped into transactions that are terminated by calls to method commits or method rollbacks. By default, new connections are in autocommit mode.	Void	SQLException	Yes
setClientInfo(Properties properties)	Sets the client information property value of the connection. The properties object contains the names and values of the client information properties to be set. The client information property set contained in the attribute list replaces the current client information property set on the connection. If the property currently set on the connection is not in the property list, the property is cleared. Currently, only ApplicationName and ApplicationType in the client are replaced.	Void	SQLClientInfoException	Yes
setClientInfo(String name,String value)	Sets the **info** property specified by **name** to the specified **value**. Currently, only ApplicationName and ApplicationType in the client are replaced.	Void	SQLClientInfoException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
setHoldability(int holdability)	Changes the default holdability of the ResultSet object created with this connection object to the given holdability. The default holdability of the ResultSet object can be determined by calling DatabaseMetaData.getResultSetHoldability.	Void	SQLException	Yes
setNetworkTimeout(Executor executor, int milliseconds)	Sets the maximum time that a connection or an object created from a connection waits for the database to reply to any request. If any request is still not answered, the waiting method returns SQLException, and the connection or object created from the connection is marked as closed. Any subsequent use of the object other than the Close, isCabled, or Connection.isValid method will result in SQLException.	Void	SQLException	Yes
setReadOnly(boolean readOnly)	Sets this connection to read-only mode as a hint for the driver to enable database optimization.	Void	SQLException	Yes
setSavepoint()	Creates an unnamed savepoint in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSavepoint(String name)	Creates a savepoint with the given name in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSchema(String schema)	Sets the given schema name to access.	Void	SQLException	Yes
setTransactionIsolation(int level)	Attempts to change the transaction isolation level of this connection object to a given level. Constants defined in the connection API are possible transaction isolation levels.	Void	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Throws	Support JDBC 4
setTypeMap(Map<String,Class<?>> map)	Installs the given TypeMap object as a type mapping for this Connection object. Such mappings enable you to map SQL structured types to driver-customized types.	Void	SQLException	Yes

NOTICE

- The autocommit mode is used by default within the API. If you disable it by running **setAutoCommit(false)**, all the statements executed later will be packaged in explicit transactions, and you cannot execute statements that cannot be executed within transactions.
- When you create a PreparedStatement class, using the RETURNING clause in the SQL statement does not take effect.

5.3.4.3 java.sql.CallableStatement

java.sql.CallableStatement is an API for executing stored procedures.

Table 5-29 Support for java.sql.CallableStatement

Method Name	Return Type	Support JDBC4
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	BLOB	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	Byte	Yes
getBytes(int parameterIndex)	Byte[]	Yes
getClob(int parameterIndex)	CLOB	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	Double	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getFloat(int parameterIndex)	Float	Yes
getInt(int parameterIndex)	Integer	Yes
getLong(int parameterIndex)	Long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getRowId(int parameterIndex)	RowId	Yes
getShort(int parameterIndex)	Short	Yes
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	Void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	Void	Yes
wasNull()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Batch operations on statements containing OUT parameters are not allowed.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.
- The following methods are inherited from java.sql.PreparedStatement: addBatch, clearParameters, execute, executeQuery, executeUpdate, getMetaData, setBigDecimal, setBoolean, setByte, setBytes, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setString, setTime, and setTimestamp.
- The registerOutParameter(int parameterIndex, int sqlType, int type) method is used only to register the composite data type.

Table 5-30 APIs supported by java.sql.CallableStatement (available only in ORA-compatible mode and when the connection parameter **oracleCompatible** is set to **functionsCompatible**)

Method Name	Return Type	Support JDBC4
setNString(String parameterName, String value)	Void	Yes
setClob(String parameterName, Reader reader, long length)	Void	Yes
setClob(String parameterName, Reader reader)	Void	Yes
setBlob(String parameterName, InputStream inputStream, long length)	Void	Yes
setBlob(String parameterName, InputStream inputStream)	Void	Yes
setBlob(String parameterName, Blob x)	Void	Yes
setClob(String parameterName, Clob x)	Void	Yes
setSQLXML(String parameterName, SQLXML xmlObject)	Void	Yes
setNull(String parameterName, int sqlType)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setBoolean(String parameterName, boolean x)	Void	Yes
setByte(String parameterName, byte x)	Void	Yes
setShort(String parameterName, short x)	Void	Yes
setInt(String parameterName, int x)	Void	Yes
setLong(String parameterName, long x)	Void	Yes
setFloat(String parameterName, float x)	Void	Yes
setDouble(String parameterName, double x)	Void	Yes
setBigDecimal(String parameterName, BigDecimal x)	Void	Yes
setString(String parameterName, String x)	Void	Yes
setBytes(String parameterName, byte[] x)	Void	Yes
setDate(String parameterName, java.sql.Date x)	Void	Yes
setTime(String parameterName, Time x)	Void	Yes
setTimestamp(String parameterName, Timestamp x)	Void	Yes
setObject(String parameterName, Object x, int targetSqlType, int scale)	Void	Yes
setObject(String parameterName, Object x, int targetSqlType)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setObject(String parameterName, Object x)	Void	Yes
setDate(String parameterName, java.sql.Date x, Calendar cal)	Void	Yes
setTime(String parameterName, Time x, Calendar cal)	Void	Yes
setTimestamp(String parameterName, Timestamp x, Calendar cal)	Void	Yes
setNull(String parameterName, int sqlType, String typeName)	Void	Yes
setRowId(String parameterName, RowId x)	Void	Yes
getObject(String parameterName)	Object	Yes
getBoolean(String parameterName)	Boolean	Yes
getByte(String parameterName)	Byte	Yes
getShort(String parameterName)	Short	Yes
getInt(String parameterName)	Int	Yes
getLong(String parameterName)	Long	Yes
getFloat(String parameterName)	Float	Yes
getDouble(String parameterName)	Double	Yes
getBytes(String parameterName)	Byte[]	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getArray(String parameterName)	Array	Yes
getDate(String parameterName)	java.sql.Date	Yes
getDate(String parameterName, Calendar cal)	java.sql.Date	Yes
getString(String parameterName)	String	Yes
getClob(String parameterName)	CLOB	Yes
getSQLXML(String parameterName)	java.sql.SQLXML	Yes
getNString(String parameterName)	String	Yes
getBigDecimal(String parameterName)	BigDecimal	Yes
getBlob(String parameterName)	BLOB	Yes
getTimestamp(String parameterName)	Timestamp	Yes
getTimestamp(String parameterName, Calendar cal)	Timestamp	Yes
getRowId(String parameterName)	RowId	Yes
execute()	Boolean	Yes
wasNull()	Boolean	Yes
registerOutParame-ter(String parameterName, int sqlType)	Void	Yes
registerOutParame-ter(String parameterName, int sqlType, int scale)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
registerOutParame-ter(String parameterName, int sqlType, String typeName)	Void	Yes

NOTE

- The connection parameter **oracleCompatible** needs to be set to **functionsCompatible**.
- A stored procedure with only one output parameter (including the return value) of the record type is not supported.
- Types.NULL, Types.OTHER, Types.JAVA_OBJECT, and Types.DISTINCT are not supported during registration.
- Only the call syntax is supported; you are advised to use indexes.
- The type conversion capability of the GET method of CallableStatement is consistent with that of the GET method of ResultSet.
- The registerOutParameter API can register **sqlType** that differs from the parameter type defined in the stored procedure.
  - When the **proc_outparam_override** parameter is disabled, all **sqlType** values can, in theory, be registered, except for the unsupported ones.
  - When the **proc_outparam_override** parameter is enabled, a type OID is bound with the registered **sqlType**. The database kernel checks whether the type supports implicit conversion to the defined type. If the function is not supported, an error is reported.

The types obtained by getObject(Int parameterIndex) and getObject(String parameterName) are irrelevant to the return value types, but are related to the registered **sqlType**. The return types corresponding to the registered **sqlType** are listed in **Table 5-31**.

Table 5-31 Return types corresponding to the registered **sqlType**

sqlType	Actual Return Type
Types.BIT	Boolean
Types.BOOLEAN	Boolean
Types.TINYINT	Byte
Types.SMALLINT	Short
Types.INTEGER	Int
Types.BIGINT	Long
Types.REAL	Float
Types.FLOAT	Double
Types.DOUBLE	Double

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

sqlType	Actual Return Type
Types.DECIMAL	BigDecimal
Types.NUMERIC	BigDecimal
Types.CHAR	String
Types.VARCHAR	String
Types.LONGVARCHAR	String
Types.DATALINK	String
Types.NCHAR	String
Types.NVARCHAR	String
Types.LONGNVARCHAR	String
Types.NCLOB	NCLOB (not supported currently)
Types.DATE	Date
Types.TIME	Time
Types.TIME_WITH_TIMEZONE	Time
Types.TIMESTAMP	Timestamp
Types.TIMESTAMP_WITH_TIMEZONE	Timestamp
Types.CLOB	CLOB
Types.BLOB	BLOB
Types.BINARY	Byte[]
Types.VARBINARY	Byte[]
Types.LONGVARBINARY	Byte[]
Types.ARRAY	Array
Types.STRUCT	Record
Types.REF	Ref (not supported currently)
Types.ROWID	RowId
Types.SQLXML	SQLXML
Types.REF_CURSOR	ResultSet

5.3.4.4 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData is an API for defining database objects.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-32 Support for java.sql.DatabaseMetaData

Method Name	Return Type	Support JDBC4
allProceduresAreCallable()	Boolean	Yes
allTablesAreSelectable()	Boolean	Yes
autoCommitFailureClosesAll-ResultSets()	Boolean	Yes
dataDefinitionCausesTran-sactionCommit()	Boolean	Yes
dataDefinitionIgnoredIn-Transactions()	Boolean	Yes
deletesAreDetected(int type)	Boolean	Yes
doesMaxRowSizeInclude-Blobs()	Boolean	Yes
generatedKeyAlwaysRe-turned()	Boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsola-tion()	Int	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	Int	Yes
getMaxCatalogName-Length()	Int	Yes
getMaxCharLiteralLength()	Int	Yes
getMaxColumnName-Length()	Int	Yes
getMaxColumnsInGroupBy()	Int	Yes
getMaxColumnsInIndex()	Int	Yes
getMaxColumnsInOrderBy()	Int	Yes
getMaxColumnsInSelect()	Int	Yes
getMaxColumnsInTable()	Int	Yes
getMaxConnections()	Int	Yes
getMaxCursorNameLength()	Int	Yes
getMaxIndexLength()	Int	Yes
getMaxLogicalLobSize()	Default long	Yes
getMaxProcedureName-Length()	Int	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getMaxRowSize()	Int	Yes
getMaxSchemaName-Length()	Int	Yes
getMaxStatementLength()	Int	Yes
getMaxStatements()	Int	Yes
getMaxTableNameLength()	Int	Yes
getMaxTablesInSelect()	Int	Yes
getMaxUserNameLength()	Int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimary-Keys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	Int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	Boolean	Yes
locatorsUpdateCopy()	Boolean	Yes
othersDeletesAreVisible(int type)	Boolean	Yes
othersInsertsAreVisible(int type)	Boolean	Yes
othersUpdatesAreVisible(int type)	Boolean	Yes
ownDeletesAreVisible(int type)	Boolean	Yes
ownInsertsAreVisible(int type)	Boolean	Yes
ownUpdatesAreVisible(int type)	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
storesMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
supportsBatchUpdates()	Boolean	Yes
supportsCatalogsInDataManipulation()	Boolean	Yes
supportsCatalogsInIndexDefinitions()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
supportsCatalogsInPrivilegeDefinitions()	Boolean	Yes
supportsCatalogsInProcedureCalls()	Boolean	Yes
supportsCatalogsInTableDefinitions()	Boolean	Yes
supportsCorrelatedSubqueries()	Boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	Boolean	Yes
supportsDataManipulationTransactionsOnly()	Boolean	Yes
supportsGetGeneratedKeys()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
supportsMultipleOpenResults()	Boolean	Yes
supportsNamedParameters()	Boolean	Yes
supportsOpenCursorsAcrossCommit()	Boolean	Yes
supportsOpenCursorsAcrossRollback()	Boolean	Yes
supportsOpenStatementsAcrossCommit()	Boolean	Yes
supportsOpenStatementsAcrossRollback()	Boolean	Yes
supportsPositionedDelete()	Boolean	Yes
supportsPositionedUpdate()	Boolean	Yes
supportsRefCursors()	Boolean	Yes
supportsResultSetConcurrency(int type, int concurrency)	Boolean	Yes
supportsResultSetType(int type)	Boolean	Yes
supportsSchemasInIndexDefinitions()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
supportsSchemasInPrivilege-Definitions()	Boolean	Yes
supportsSchemasInProce-dureCalls()	Boolean	Yes
supportsSchemasInTableDe-finitions()	Boolean	Yes
supportsSelectForUpdate()	Boolean	Yes
supportsStatementPooling()	Boolean	Yes
supportsStoredFunctionsU-singCallSyntax()	Boolean	Yes
supportsStoredProcedures()	Boolean	Yes
supportsSubqueriesInCom-parisons()	Boolean	Yes
supportsSubqueriesInExists()	Boolean	Yes
supportsSubqueriesInIns()	Boolean	Yes
supportsSubqueriesInQuan-tifieds()	Boolean	Yes
supportsTransactionIsola-tionLevel(int level)	Boolean	Yes
supportsTransactions()	Boolean	Yes
supportsUnion()	Boolean	Yes
supportsUnionAll()	Boolean	Yes
updatesAreDetected(int type)	Boolean	Yes
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	Boolean	Yes
nullsAreSortedHigh()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
nullsAreSortedLow()	Boolean	Yes
nullsAreSortedAtStart()	Boolean	Yes
nullsAreSortedAtEnd()	Boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	Int	Yes
getDriverMinorVersion()	Int	Yes
usesLocalFiles()	Boolean	Yes
usesLocalFilePerTable()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
supportsMixedCaseQuotedIdentifiers()	Boolean	Yes
storesUpperCaseQuotedIdentifiers()	Boolean	Yes
storesLowerCaseQuotedIdentifiers()	Boolean	Yes
storesMixedCaseQuotedIdentifiers()	Boolean	Yes
supportsAlterTableWithAddColumn()	Boolean	Yes
supportsAlterTableWithDropColumn()	Boolean	Yes
supportsColumnAliasing()	Boolean	Yes
nullPlusNonNullIsNull()	Boolean	Yes
supportsConvert()	Boolean	Yes
supportsConvert(int fromType, int toType)	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
supportsTableCorrelation-Names()	Boolean	Yes
supportsDifferentTableCorre-lationNames()	Boolean	Yes
supportsExpressionsInOrder-By()	Boolean	Yes
supportsOrderByUnrelated()	Boolean	Yes
supportsGroupBy()	Boolean	Yes
supportsGroupByUnrelated()	Boolean	Yes
supportsGroupByBeyondSe-lect()	Boolean	Yes
supportsLikeEscapeClause()	Boolean	Yes
supportsMultipleResultSets()	Boolean	Yes
supportsMultipleTransac-tions()	Boolean	Yes
supportsNonNullableCol-umns()	Boolean	Yes
supportsMinimumSQLGram-mar()	Boolean	Yes
supportsCoreSQLGrammar()	Boolean	Yes
supportsExtendedSQLGram-mar()	Boolean	Yes
supportsANSI92EntryLevelS QL()	Boolean	Yes
supportsANSI92Intermediate SQL()	Boolean	Yes
supportsANSI92FullSQL()	Boolean	Yes
supportsIntegrityEnhance-mentFacility()	Boolean	Yes
supportsOuterJoins()	Boolean	Yes
supportsFullOuterJoins()	Boolean	Yes
supportsLimitedOuterJoins()	Boolean	Yes
isCatalogAtStart()	Boolean	Yes
supportsSchemasInDataMa-nipulation()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
supportsSavepoints()	Boolean	Yes
supportsResultSetHoldability(int holdability)	Boolean	Yes
getResultSetHoldability()	Int	Yes
getDatabaseMajorVersion()	Int	Yes
getDatabaseMinorVersion()	Int	Yes
getJDBCMajorVersion()	Int	Yes
getJDBCMinorVersion()	Int	Yes

NOTE

- When **uppercaseAttributeName** is **true**, the following methods convert query results to uppercase, all within the range defined by Java's toUpperCase method.
  - public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
  - public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
  - public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
  - public ResultSet getSchemas(String catalog, String schemaPattern)
  - public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
  - public ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
  - public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
  - public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
  - public ResultSet getPrimaryKeys(String catalog, String schema, String table)
  - protected ResultSet getImportedExportedKeys(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
  - public ResultSet getIndexInfo(String catalog, String schema, String tableName, boolean unique, boolean approximate)
  - public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
  - public ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)
- The getPartitionTablePrimaryKeys(String catalog, String schema, String table) method retrieves the primary key columns of a partitioned table with global indexes. Here is an example:

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.5 java.sql.Driver

java.sql.Driver is a database driver API.

Table 5-33 Support for java.sql.Driver

Method Name	Return Type	Support JDBC4
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	Boolean	Yes
getMajorVersion()	Int	Yes
getMinorVersion()	Int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.3.4.6 java.sql.PreparedStatement

java.sql.PreparedStatement is a prepared statement API.

Table 5-34 Support for java.sql.PreparedStatement

Method Name	Return Type	Support JDBC4
clearParameters()	Void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
executeUpdate()	Int	Yes
executeLargeUpdate()	Long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	Void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setAsciiStream(int parameterIndex, InputStream x, long length)	Void	Yes
setAsciiStream(int parameterIndex, InputStream x)	Void	Yes
setBinaryStream(int parameterIndex, InputStream x)	Void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	Void	Yes
setBinaryStream(int parameterIndex, InputStream x, long length)	Void	Yes
setBlob(int parameterIndex, InputStream inputStream)	Void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	Void	Yes
setBlob(int parameterIndex, Blob x)	Void	Yes
setCharacterStream(int parameterIndex, Reader reader)	Void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	Void	Yes
setCharacterStream(int parameterIndex, Reader value, long length)	Void	Yes
setClob(int parameterIndex, Reader reader)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setClob(int parameterIndex, Reader reader, long length)	Void	Yes
setClob(int parameterIndex, Clob x)	Void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	Void	Yes
setNull(int parameterIndex, int sqlType)	Void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	Void	Yes
setObject(int parameterIndex, Object x)	Void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	Void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	Void	Yes
setObject(int parameterIndex, Object x, SQLType targetSqlType, int scaleOrLength)	Void	Yes
setObject(int parameterIndex, Object x, SQLType targetSqlType)	Void	Yes
setRowId(int parameterIndex, RowId x)	Void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	Void	Yes
setTime(int parameterIndex, Time x)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setTime(int parameterIndex, Time x, Calendar cal)	Void	Yes
setTimestamp(int parameterIndex, Timestamp x)	Void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	Void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	Void	Yes
setURL(int parameterIndex, URL x)	Void	Yes
setBoolean(int parameterIndex, boolean x)	Void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	Void	Yes
setByte(int parameterIndex, byte x)	Void	Yes
setBytes(int parameterIndex, byte[] x)	Void	Yes
setDate(int parameterIndex, Date x)	Void	Yes
setDouble(int parameterIndex, double x)	Void	Yes
setFloat(int parameterIndex, float x)	Void	Yes
setInt(int parameterIndex, int x)	Void	Yes
setLong(int parameterIndex, long x)	Void	Yes
setShort(int parameterIndex, short x)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setString(int parameterIndex, String x)	Void	Yes
setNString(int parameterIndex, String x)	Void	Yes
addBatch()	Void	Yes
executeBatch()	Int[]	Yes

NOTE

- Execute **addBatch()** and **execute()** only after running **clearBatch()**.
- The executeBatch() method does not clear batches. To clear batches, you must explicitly call clearBatch().
- After adding a batch of bound variables, you can reuse their values for subsequent batches without calling the set*() method again.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, setFetchSize, and enableStreamingResults.
- The executeLargeUpdate() method can only be used in JDBC 4.2 or later.
- When an SQL statement is executed, the maximum size of a packet cannot exceed 1023 MB. Therefore, when using methods such as setAsciiStream and setCharacterStream, you need to evaluate and control the data size to avoid exceptions caused by oversized packets.
- During batch insertion using the executeBatch() method, an exception in any data item will result in a full rollback of the entire batch because partial commits are not supported.

5.3.4.7 java.sql.ResultSet

java.sql.ResultSet is an execution result set API.

Table 5-35 Support for java.sql.ResultSet

Method Name	Return Type	Support JDBC4
absolute(int row)	Boolean	Yes
afterLast()	Void	Yes
beforeFirst()	Void	Yes
cancelRowUpdates()	Void	Yes
clearWarnings()	Void	Yes
close()	Void	Yes
deleteRow()	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
findColumn(String columnLabel)	Int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	BLOB	Yes
getBlob(String columnLabel)	BLOB	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	Byte	Yes
getBytes(int columnIndex)	Byte[]	Yes
getByte(String columnLabel)	Byte	Yes
getBytes(String columnLabel)	Byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream(String columnLabel)	Reader	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getClob(int columnIndex)	CLOB	Yes
getClob(String columnLabel)	CLOB	Yes
getConcurrency()	Int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes
getDouble(int columnIndex)	Double	Yes
getDouble(String columnLabel)	Double	Yes
getFetchDirection()	Int	Yes
getFetchSize()	Int	Yes
getFloat(int columnIndex)	Float	Yes
getFloat(String columnLabel)	Float	Yes
getInt(int columnIndex)	Int	Yes
getInt(String columnLabel)	Int	Yes
getLong(int columnIndex)	Long	Yes
getLong(String columnLabel)	Long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getObject(int columnIndex, Map<String,Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String,Class<?>> map)	Object	Yes
getRow()	Int	Yes
getRowId(int columnIndex)	RowId	Yes
getRowId(String columnLabel)	RowId	Yes
getShort(int columnIndex)	Short	Yes
getShort(String columnLabel)	Short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	Int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	Void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	Void	Yes
moveToInsertRow()	Void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	Void	Yes
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
setFetchDirection(int direction)	Void	Yes
setFetchSize(int rows)	Void	Yes
updateArray(int columnIndex, Array x)	Void	Yes
updateArray(String columnLabel, Array x)	Void	Yes
updateAsciiStream(int columnIndex, InputStream x)	Void	Yes
updateAsciiStream(String columnLabel, InputStream x)	Void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	Void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	Void	Yes
updateAsciiStream(int columnIndex, InputStream x)	Void	Yes
updateAsciiStream(String columnLabel, InputStream x)	Void	Yes
updateBigDecimal(int columnIndex, BigDecimal x)	Void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	Void	Yes
updateBinaryStream(int columnIndex, InputStream x)	Void	Yes
updateBinaryStream(Strin g columnLabel, InputStream x)	Void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
updateBinaryStream(String columnLabel, InputStream x, int length)	Void	Yes
updateBinaryStream(int columnIndex, InputStream x, long length)	Void	Yes
updateBinaryStream(String columnLabel, InputStream x, long length)	Void	Yes
updateBlob(int columnIndex, Blob x)	Void	Yes
updateBlob(String columnLabel, Blob x)	Void	Yes
updateBlob(int columnIndex, InputStream x)	Void	Yes
updateBlob(String columnLabel, InputStream x)	Void	Yes
updateBlob(int columnIndex, InputStream x, long length)	Void	Yes
updateBlob(String columnLabel, InputStream x, long length)	Void	Yes
updateBoolean(int columnIndex, boolean x)	Void	Yes
updateBoolean(String columnLabel, boolean x)	Void	Yes
updateByte(int columnIndex, byte x)	Void	Yes
updateByte(String columnLabel, byte x)	Void	Yes
updateBytes(int columnIndex, byte[] x)	Void	Yes
updateBytes(String columnLabel, byte[] x)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
updateCharacter-Stream(int columnIndex, Reader x)	Void	Yes
updateCharacter-Stream(String columnLabel, Reader x)	Void	Yes
updateCharacter-Stream(int columnIndex, Reader x, int length)	Void	Yes
updateCharacter-Stream(String columnLabel, Reader reader, int length)	Void	Yes
updateCharacter-Stream(int columnIndex, Reader x, long length)	Void	Yes
updateCharacter-Stream(String columnLabel, Reader x, long length)	Void	Yes
updateClob(int columnIndex, Clob x)	Void	Yes
updateClob(String columnLabel, Clob x)	Void	Yes
updateClob(int columnIndex, Reader x)	Void	Yes
updateClob(String columnLabel, Reader x)	Void	Yes
updateClob(int columnIndex, Reader x, long length)	Void	Yes
updateClob(String columnLabel, Reader x, long length)	Void	Yes
updateDate(int columnIndex, Date x)	Void	Yes
updateDate(String columnLabel, Date x)	Void	Yes
updateDouble(int columnIndex, double x)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
updateDouble(String columnLabel, double x)	Void	Yes
updateFloat(int columnIndex, float x)	Void	Yes
updateFloat(String columnLabel, float x)	Void	Yes
updateInt(int columnIndex, int x)	Void	Yes
updateInt(String columnLabel, int x)	Void	Yes
updateLong(int columnIndex, long x)	Void	Yes
updateLong(String columnLabel, long x)	Void	Yes
updateNull(int columnIndex)	Void	Yes
updateNull(String columnLabel)	Void	Yes
updateObject(int columnIndex, Object x)	Void	Yes
updateObject(int columnIndex, Object x, int scaleOrLength)	Void	Yes
updateObject(String columnLabel, Object x)	Void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	Void	Yes
updateRow()	Void	Yes
updateRowId(int columnIndex, RowId x)	Void	Yes
updateRowId(String columnLabel, RowId x)	Void	Yes
updateShort(int columnIndex, short x)	Void	Yes
updateShort(String columnLabel, short x)	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
updateSQLXML(int columnIndex, SQLXML xmlObject)	Void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	Void	Yes
updateString(int columnIndex, String x)	Void	Yes
updateString(String columnLabel, String x)	Void	Yes
updateTime(int columnIndex, Time x)	Void	Yes
updateTime(String columnLabel, Time x)	Void	Yes
updateTimestamp(int columnIndex, Timestamp x)	Void	Yes
updateTimestamp(String columnLabel, Timestamp x)	Void	Yes
wasNull()	Boolean	Yes

NOTE

- One statement cannot have multiple open ResultSets.
- The cursor used for traversing the ResultSet cannot remain open after being committed.
- When an SQL statement is executed, the packet size cannot exceed 1,023 MB. Therefore, when using methods such as updateAsciiStream, updateBlob, updateBinaryStream, updateClob, and updateCharacterStream, you need to evaluate and control the data size to avoid exceptions caused by oversized packets.
- APIs related to updateRow() can update only tables with a primary key.

5.3.4.8 java.sql.ResultSetMetaData

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

Table 5-36 Support for java.sql.ResultSetMetaData

Method Name	Return Type	Support JDBC4
getCatalogName(int column)	String	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getColumnClassName(int column)	String	Yes
getColumnCount()	Int	Yes
getColumnDisplaySize(int column)	Int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	Int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	Int	Yes
getScale(int column)	Int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	Boolean	Yes
isCaseSensitive(int column)	Boolean	Yes
isCurrency(int column)	Boolean	Yes
isDefinitelyWritable(int column)	Boolean	Yes
isNullable(int column)	Int	Yes
isReadOnly(int column)	Boolean	Yes
isSearchable(int column)	Boolean	Yes
isSigned(int column)	Boolean	Yes
isWritable(int column)	Boolean	Yes

NOTE

When **uppercaseAttributeName** is set to **true**, the following APIs will convert the query results to uppercase, encompassing all 26 English letters:

- public String getColumnName(int column)
- public String getColumnLabel(int column)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.9 java.sql.Statement

java.sql.Statement is an SQL statement API.

Table 5-37 Support for java.sql.Statement

Method Name	Return Type	Support JDBC4
addBatch(String sql)	Void	Yes
clearBatch()	Void	Yes
clearWarnings()	Void	Yes
close()	Void	Yes
closeOnCompletion()	Void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	Int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	Int	Yes
executeUpdate(String sql, int[] columnIndexes)	Int	Yes
executeUpdate(String sql, String[] columnNames)	Int	Yes
getConnection()	Connection	Yes
getFetchDirection()	Int	Yes
getFetchSize()	Int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	Int	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
getMaxRows()	Int	Yes
getLargeMaxRows()	Long	Yes
getMoreResults()	Boolean	Yes
getMoreResults(int current)	Boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrency()	Int	Yes
getResultSetHoldability()	Int	Yes
getResultSetType()	Int	Yes
getQueryTimeout()	Int	Yes
getUpdateCount()	Int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	Void	Yes
setEscapeProcessing(boolean enable)	Void	Yes
setFetchDirection(int direction)	Void	Yes
setMaxFieldSize(int max)	Void	Yes
setMaxRows(int max)	Void	Yes
setLargeMaxRows(long max)	Void	Yes
setPoolable(boolean poolable)	Void	Yes
setQueryTimeout(int seconds)	Void	Yes
setFetchSize(int rows)	Void	Yes
cancel()	Void	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
executeLargeUpdate(String sql)	Long	No
getLargeUpdateCount()	Long	No
executeLargeBatch()	Long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	Long	No
executeLargeUpdate(String sql, int[] columnIndexes)	Long	No
executeLargeUpdate(String sql, String[] columnNames)	Long	No
enableStreamingResults()	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 increases communication traffic between the database and the client and affects performance.
- Database cursors are valid only within their transactions. If **setFetchSize** is set, you need to set the connection to the non-autocommit mode by setting **setAutoCommit(false)**. and commit transactions on the connection to flush service data to a database.
- The LargeUpdate method can only be used in JDBC 4.2 or later.
- enableStreamingResults() is a customized API for enabling streaming read. This API indirectly calls setFetchSize(Integer.MIN_VALUE). To enable the streaming read function, set **enableStreamingQuery** in the URL to **true** and call setFetchSize(Integer.MIN_VALUE) or enableStreamingResults(). Except that the streaming read function is enabled, the input parameter of setFetchSize() can only be a positive number or **0**.
- The executeLargeUpdate method is supported when M compatibility is enabled, that is, the connection parameter **mCompatible** is set to **true**. For details on enabling M compatibility, see **Connection Parameter Reference**.
- Currently, the setLargeMaxRows method does not accept parameter values exceeding **Integer.MAX_VALUE**.

5.3.4.10 java.sql.Clob

The java.sql.Clob API is used to obtain SQL character large objects and access SQL CLOB values.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-38 Common methods of using java.sql.Clob

Method Name	Return Value	Description	Throws
length()	Long	Returns the number of CLOB characters.	SQLException
getSubString(long pos, int length)	String	Returns a string of consecutive characters with length specified by **length** and starting from position specified by **pos**.	SQLException
getCharacterStream()	java.io.Reader	Returns a string stream containing the CLOB value.	SQLException
getAsciiStream()	java.io.InputStream	Returns an ASCII stream containing the CLOB value. **NOTICE** Currently, this method is not supported.	SQLException
position(String searchstr, long start)	Long	Retrieves a specified character string. The retrieval starts from the position specified by **start** and returns the first position at which the character string appears.	SQLException
position(Clob searchstr, long start)	Long	Retrieves a specified CLOB. The retrieval starts from the position specified by **start** and returns the first position at which the CLOB appears.	SQLException
setString(long pos, String str)	Int	Writes **str** to a CLOB starting from the position specified by **pos**.	SQLException
setString(long pos, String str, int offset, int len)	Int	Writes **len** characters of **str** (starting at character **offset**) to a CLOB starting from the position specified by **pos**.	SQLException

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Value	Description	Throws
setAsciiStream(long pos)	java.io.OutputStream	Returns an ASCII stream that starts from the position specified by **pos** and can be written to a CLOB. **NOTICE** Currently, this method is not supported.	SQLException
setCharacterStream(long pos)	java.io.Writer	Returns a character stream that starts from the position specified by **pos** and can be written to a CLOB. **NOTICE** Currently, this method is not supported.	SQLException
truncate(long len)	Void	Truncates a CLOB string of the length specified by **len**.	SQLException
free()	Void	Frees the CLOB and releases resources.	SQLException
getCharacterStream(long pos, long length)	Reader	Returns a Reader object that contains a partial CLOB value, which starts from the character specified by **pos** and has the length specified by **length**.	SQLException

5.3.4.11 java.sql.Array

The java.sql.Array API is the JDBC mapping of SQL collection types. It provides methods for transmitting SQL data to clients as an array or ResultSet object.

Table 5-39 Common methods of using java.sql.Array

Method Name	Return Type	throws	Support JDBC4
getBaseTypeName()	String	SQLException	Yes
getBaseType()	int	SQLException	Yes
getArray()	Object	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	throws	Support JDBC4
getArray(java.util.Map<String,Class<?>> map)	Object	SQLException	No
getArray(long index, int count)	Object	SQLException	Yes
getArray(long index, int count, java.util.Map<String,Class<?>> map)	Object	SQLException	No
getResultSet()	ResultSet	SQLException	No
getResultSet(java.util.Map<String,Class<?>> map)	ResultSet	SQLException	No
getResultSet(long index, int count)	ResultSet	SQLException	No
getResultSet (long index, int count, java.util.Map<String,Class<?>> map)	ResultSet	SQLException	No
free()	void	SQLException	No

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- For the array types obtained by the getArray and getArray(long index, int count) APIs, refer to the **Java Variable Type** column in **Data Type Mapping**.

  For example, if the element type is varchar, the object type obtained by the getArray API is String[].

  Special notes: For an element of the int2 or smallint type, if the element type is changed to short when the element is accessed, the object type obtained by the getArray API is Short[].
- For **sqlType** obtained by the getBaseType API, refer to the **JDBC Type Index** column in **Data Type Mapping**.
- The element type names returned by the getBaseTypeName API are outlined in **Data Type Mapping**. Here are the mapping rules for the collection, array, and record type names:
  1. If a type is defined in a package, its name is generally in *schema.package.type* format.

     If any name in *schema*, *package*, or *type* violates naming conventions, the type name will follow the *"schema"."package"."type"* format.

     - Only letters, digits, and underscores (_) are allowed.
     - The first character cannot be a digit.
  2. If a type is defined in a schema, its name is generally in *schema.type* format.

     If any of schema and type names does not comply with the rules, the type name is in the **"***Schema***"."***Type***"** format.

     - Only letters, digits, and underscores (_) are allowed.
     - The first character cannot be a digit.
- For the getArray(long index, int count) API:
  1. The value of **index** is 1 for the start element.
  2. The value of **index** ranges from 1 to 2147483647. If the input value is out of this range, an error is reported.
  3. The value of **count** ranges from 0 to 2147483647. If the input value is out of this range, an error is reported.
  4. If the value of **index** exceeds the number of elements, an empty array whose length is 0 is returned.
  5. If the value of **count** exceeds the maximum number of elements that can be obtained based on the current index, an array consisting of the remaining elements is returned.

5.3.4.12 java.sql.Struct

The java.sql.Struct API is the JDBC mapping of SQL structured types, where a Struct object contains attribute values of its corresponding SQL structured type. This API provides methods for transmitting SQL data to clients as an array.

Table 5-40 Common methods of using java.sql.Struct

Method Name	Return Type	throws	Support JDBC4
getSQLTypeName()	String	SQLException	Yes
getAttributes()	Object[]	SQLException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	throws	Support JDBC4
getAttributes(java.util.Map<String,Class<?>> map)	Object[]	SQLException	No

NOTE

- The object type obtained by the getAttribute API is Object[]. For details about the type of each element in the returned array, see the **Java Variable Type** column in **Data Type Mapping**.

  For example, for the record(c1 int, c2 varchar(5)) type, the first element in Object[] returned by getAttribute is **null** or an integer, and the second element is **null** or a string.

  Special notes: For an element of the int2 or smallint type, the element type is changed to short when the element is accessed.

- The type names returned by the getSQLTypeName API are outlined in **Data Type Mapping**. Here are the mapping rules for the collection, array, and record type names:

  1. If a type is defined in a package, its name is generally in *schema.package.type* format and must comply with the following conventions:
     - Only letters, digits, and underscores (_) are allowed.
     - The first character cannot be a digit.

     If any name in *schema*, *package*, or *type* violates naming conventions, double quotation marks will be required, that is, *"schema"."package"."type"*.
  2. If a type is defined in a schema, its name is generally in *schema.type* format and must comply with the following conventions:
     - Only letters, digits, and underscores (_) are allowed.
     - The first character cannot be a digit.

     If any name in *schema* or *type* violates naming conventions, double quotation marks will be required, that is, *"schema"."type"*.

5.3.4.13 java.sql.SQLException

java.sql.SQLException represents a core exception class for database operations through JDBC APIs. This exception is thrown when an error occurs during JDBC operations (such as executing SQL statements or processing result sets).

Table 5-41 Support for java.sql.SQLException

Method Name	Description	Return Type	Support JDBC4
getMessage()	Returns the description of the error.	String	Yes
getSQLState()	Returns the standard SQL status code.	String	Yes
getErrorCode()	Returns a specific error code.	Int	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Description	Return Type	Support JDBC4
getCause()	Returns the root cause of this exception.	Throwable	Yes
getNextException()	Returns the next SQL exception associated with this exception (needed for processing an exception chain).	SQLException	Yes
setNextException(SQLException ex)	Links exceptions to the end of the current exception chain.	Void	Yes
printStackTrace()	Prints the stack information of the current exception to the standard error stream.	Void	Yes

5.3.4.14 java.io.IOException

java.io.IOException represents a core checked exception class for Java I/O operations. It is used to indicate an error or interruption during input/output operations.

Table 5-42 Support for java.io.IOException

Method Name	Description	Return Type	Support JDBC4
getMessage()	Returns details about the error.	String	Yes
getCause()	Returns the root cause of this exception.	Throwable	Yes
printStackTrace()	Prints the stack information of the current exception to the standard error stream.	Void	Yes
initCause(Throwable cause)	Sets the cause of this exception (typically used to wrap underlying exceptions).	Throwable	Yes
toString()	Returns a string that contains the class name and error information of this exception.	String	Yes
getSuppressed()	Returns the array of all suppressed exceptions.	Throwable[]	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.15 javax.sql.ConnectionPoolDataSource

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

Table 5-43 Support for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	Support JDBC4
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

5.3.4.16 javax.sql.DataSource

javax.sql.DataSource is a data source API.

Table 5-44 Support for javax.sql.DataSource

Method Name	Return Type	Support JDBC4
getConnection()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	Int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	Void	Yes
setLogWriter(PrintWriter out)	Void	Yes

5.3.4.17 javax.sql.PooledConnection

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

Table 5-45 Support for javax.sql.PooledConnection

Method Name	Return Type	Support JDBC4
addConnectionEventListener (ConnectionEventListener listener)	Void	Yes
close()	Void	Yes
getConnection()	Connection	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Support JDBC4
removeConnectionEventListener (ConnectionEventListener listener)	Void	Yes

5.3.4.18 javax.naming.Context

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

Table 5-46 Support for javax.naming.Context

Method Name	Return Type	Support JDBC4
bind(Name name, Object obj)	Void	Yes
bind(String name, Object obj)	Void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	Void	Yes
rebind(String name, Object obj)	Void	Yes
rename(Name oldName, Name newName)	Void	Yes
rename(String oldName, String newName)	Void	Yes
unbind(Name name)	Void	Yes
unbind(String name)	Void	Yes

5.3.4.19 CopyManager

CopyManager is an API class provided by GaussDB's JDBC driver. It implements batch data import/export for databases.

Inheritance Relationship of CopyManager

The CopyManager class is in the **com.huawei.gaussdb.jdbc.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Constructor

public CopyManager(BaseConnection connection)

throws SQLException

Common Methods

Table 5-47 Common methods of CopyManager

Method Name	Return Type	Description	Throws	Support JDBC4
copyIn(String sql)	CopyIn	-	SQLException	Yes
copyIn(String sql, InputStream from)	Long	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, InputStream from, int bufferSize)	Long	Uses **COPY FROM STDIN** to quickly load data of a specified length to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, Reader from)	Long	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException	Yes
copyIn(String sql, Reader from, int bufferSize)	Long	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException	Yes
copyOut(String sql)	CopyOut	-	SQLException	Yes
copyOut(String sql, OutputStream to)	Long	Sends the result set of **COPY TO STDOUT** from the database to the OutputStream class.	SQLException,IOException	Yes
copyOut(String sql, Writer to)	Long	Sends the result set of **COPY TO STDOUT** from the database to the Writer class.	SQLException,IOException	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.20 PGReplicationConnection

PGReplicationConnection is an API class provided by the JDBC driver in GaussDB. It is used to implement functions related to logical replication.

Inheritance Relationship of PGReplicationConnection

PGReplicationConnection is a logical replication API. In non-multi-primary mode, its implementation class is PGReplicationConnectionImpl, located in the **com.huawei.gaussdb.jdbc.replication** package. This class is declared as follows:

```
public class PGReplicationConnectionImpl implements PGReplicationConnection
```

Constructor Method

- public PGReplicationConnectionImpl(BaseConnection connection)
- public TenantReplicationConnectionImpl(BaseConnection connection)

Common Methods

Table 5-48 Common methods of PGReplicationConnection

Method Name	Return Type	Description	Throws
createReplicationSlot()	ChainedCreateReplicationSlotBuilder	Creates a logical replication slot. Only cluster-level (CSN-based) logical replication slots can be created to connect to CNs, and replication slots with the same name can be created on other CNs and on primary DNs. Only local (LSN-based) logical replication slots can be created to connect to DNs. For details about how to create an LSN-based logical replication slot on a CN or a CSN-based logical replication slot on a DN, see the SQL function pg_create_logical_replication_slot for logical replication.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Description	Throws
dropReplicationSlot(String slotName)	Void	Deletes a logical replication slot. If the replication slot you wish to delete from a CN is LSN-based, it will be deleted only from the current node, leaving same-named slots on other nodes unaffected (if any). Otherwise, as long as other nodes have same-named CSN-based logical replication slots, the deletion will not trigger an error even if the replication slot is not found on some nodes; instead, all same-named slots will be deleted from all nodes. However, if the replication slot is not found on any node, an error will be reported. If an LSN-based logical replication slot exists on the current CN and a same-named CSN-based logical replication slot exists on other nodes, deleting the replication slot on the current CN will only remove the local LSN-based logical replication slot. After the deletion is complete, repeat the deletion operation to remove the same-named replication slot from other nodes.	SQLException,IOException
replicationStream()	ChainedStreamBuilder	Enables logical replication.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.3.4.21 PGReplicationStream

PGReplicationStream is an API class provided by the GaussDB JDBC driver. It is used to operate on logical replication streams.

Inheritance Relationship of PGReplicationStream

PGReplicationStream is a logical replication API. In non-multi-primary mode, its implementation class is V3PGReplicationStream, located in the **com.huawei.gaussdb.jdbc.core.v3.replication** package. This class is declared as follows:

```
public class V3PGReplicationStream implements PGReplicationStream
```

Constructor

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

Common Methods

Table 5-49 Common methods of PGReplicationStream

Method Name	Return Type	Description	Throws
close()	Void	Ends the logical replication and releases resources.	SQLException
forceUpdateStatus()	Void	Forcibly sends the LSN status received, refreshed, and applied last time to the backend.	SQLException
getLastAppliedLSN()	LogSequenceNumber	Retrieves the LSN when the primary node replayed logs last time.	-
getLastFlushedLSN()	LogSequenceNumber	Retrieves the LSN flushed by the primary node last time, that is, the LSN pushed by the current logical decoding.	-
getLastReceiveLSN()	LogSequenceNumber	Retrieves the last received LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots).	-
isClosed()	Boolean	Determines whether the replication stream is disabled.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Description	Throws
read()	ByteBuffer	Reads the next WAL record from the backend. If the data cannot be read, this method blocks the I/O read.	SQLException
readPending()	ByteBuffer	Reads the next WAL record from the backend. If the data cannot be read, this method does not block the I/O read.	SQLException
setAppliedLSN(LogSequenceNumber applied)	Void	Sets the applied LSN.	-
setFlushedLSN(LogSequenceNumber flushed)	Void	Updates the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots), which is sent to the backend at the next update to update the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots) on the server.	-

5.3.4.22 ChainedStreamBuilder

ChainedStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to build replication streams. This API class is supported only in non-multi-primary mode.

Inheritance Relationship of ChainedStreamBuilder

ChainedStreamBuilder is a logical replication API. Its implementation class is ReplicationStreamBuilder, which is in the **com.huawei.gaussdb.jdbc.replication.fluent** package. The declaration of the class is as follows:

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

Constructor

public ReplicationStreamBuilder(final BaseConnection connection)

Common Methods

Table 5-50 Common methods of ReplicationStreamBuilder

Method Name	Return Type	Description	Throws
logical()	ChainedLogicalStreamBuilder	Creates a logical replication stream.	-
physical()	ChainedPhysical-StreamBuilder	Creates a physical replication stream.	-

5.3.4.23 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to specify common parameters for logical and physical replication. This API class is supported only in non-multi-primary mode.

Inheritance Relationship of ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API for logical replication. The implementation abstract class is AbstractCreateSlotBuilder. The inheritance class is LogicalCreateSlotBuilder which is in the **com.huawei.gaussdb.jdbc.replication.fluent.logical** package. The declaration of this class is as follows:

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

Constructor

public LogicalCreateSlotBuilder(BaseConnection connection)

Common Methods

Table 5-51 Common methods of LogicalCreateSlotBuilder

Method Name	Return Type	Description	Throws
withSlotName(String slotName)	T	Specifies the name of a replication slot.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Description	Throws
withOutputPlugin(String outputPlugin)	ChainedLogicalCreateSlotBuilder	Specifies a plug-in name. Currently, **mppdb_decoding**, **sql_decoding**, **parallel_binary_decoding**, **parallel_json_decoding**, and **parallel_text_decoding** are supported. **mppdb_decoding**: a decoding output format. After this value is set, the output content is in JSON format. The output result contains the attribute information and attribute values of the related data. **sql_decoding**, **parallel_binary_decoding**, **parallel_json_decoding**, and **parallel_text_decoding** are output in SQL, binary, JSON, and text format, respectively, based on the plug-in format.	-
make()	Void	Creates a slot with the specified parameters in the database.	SQLException
self()	ChainedLogicalCreateSlotBuilder	Returns the implementation of ChainedLogicalCreateSlot-Builder.	-

5.3.4.24 PGobject

The PGobject class represents types that are not defined by the JDBC standards.

Table 5-52 Common PGobject methods

Method Name	Return Type	Description	Throws	Support JDBC4
getStruct()	Object[]	Obtains the subtype names of the composite type and sorts them according to the creation sequence.	-	Yes
getValue()	String	Obtains the string value of the composite type.	-	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method Name	Return Type	Description	Throws	Support JDBC4
getArrayValue()	String[]	Obtains the array values of the composite type and sorts the values according to the sequence of columns of the composite type.	-	Yes
getAttributes()	Object[]	Obtains the array values of the composite type. If the columns are of the table or array type, PgArray is returned. If the columns are of the composite type, PGobject is returned. For other types, a string value is returned.	SQLException	Yes

5.3.4.25 LogSequenceNumber

The LogSequenceNumber class manages log sequence numbers (LSNs) for logical decoding scenarios. Its *LogSequenceValue* variable uniquely identifies the encoded log corresponding to each SQL statement.

In non-multi-primary mode, the subclass is CommonLogSequenceValue.

Table 5-53 Common methods of LogSequenceNumber

Method Name	Return Type	Description
ValueOf(long value)	LogSequenceNumber	Obtains the LogSequenceNumber object in non-multi-primary mode based on the input long-type LSN value.
VlueOf(String strValue)	LogSequenceNumber	Obtains the LogSequenceNumber object based on the input string.
asLong()	long	Obtains the LSN value in a log sequence.
asLongArray()	long	Returns sequence numbers as an array. In non-multi-primary mode, LSN values are returned.
compareTo(LogSequenceNumber anotherLSN)	int	Compares the *LogSequenceValue* of two LogSequenceNumber objects. In non-multi-primary mode, LSN values are compared. If the current *LogSequenceValue* is greater than the one in the input class, **1** is returned. If they are equal, **0** is returned. In other scenarios, **-1** is returned.
equals(Object o)	boolean	Returns **true** if LSN values are equal in non-multi-primary mode; returns **false** if the input is null, the input class is not LogSequenceNumber, or *LogSequenceValue* in the class is different.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

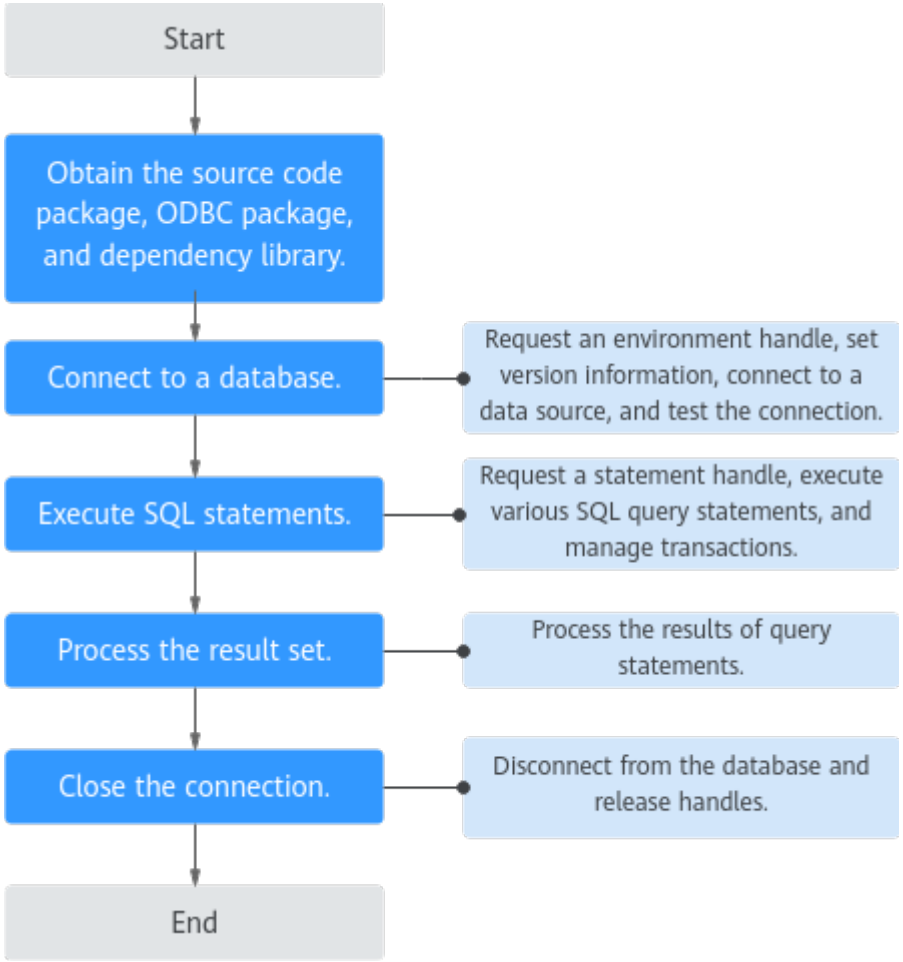
5.4 Development Based on ODBC

5.4.1 Development Process

Figure 5-13 shows the development process based on ODBC.

This process involves five steps: obtain the source code package, ODBC package, and dependency library, connect to a database, execute SQL statements, process the result set, and close the connection.

Figure 5-13 Development process based on ODBC



5.4.2 Development Procedure

5.4.2.1 Obtaining the Source Code Package, ODBC Package, and Dependency Library

To develop applications based on ODBC, you need to obtain the unixODBC source code package, ODBC package, and dependency library.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Obtaining the Source Code Package

- To obtain the unixODBC source code package, visit **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**.
- To obtain the MD5 file, visit **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**.

Use the MD5 file to verify the integrity of the unixODBC source code package. (Check whether the MD5 value is the same as that in the source code package.)

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 5-54**.

Table 5-54 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

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

- Verifying the software package integrity on a Linux server:
  a. Upload the software package and verification package to the same directory on the VM.
  b. Run the following command to verify the integrity of the software package:
     ```
     F=$(echo DBS-GaussDB-driver*.tar.gz);echo "$(awk '{print $1}' "$F.sha256")  $F" | sha256sum -c -
     ```
     If **OK** is displayed in the command output, the verification is successful. Example:
     ```
     DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz: OK
     ```
- Verifying the software package integrity on a Windows server:
  a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
  b. Run the following command to obtain the hash value of the driver package:

     certutil -hashfile *{local_directory_of_the_driver_package}**{driver_package_name}* **sha256**

     - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
     - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz**.

     Example: **certutil -hashfile** *C:|Users**DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz* **sha256**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-54**.

- If they are consistent, the verification is successful.
- If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining the ODBC Package and Dependency Library

The ODBC package and dependency library are specific to Linux and Windows OSs. Obtain the appropriate ones based on your OS.

- Linux OS:

  Obtain the ODBC package from the driver package. Different variants of the driver package are provided for various database kernel OSs. Select the one matching your actual database kernel OS. The file hierarchy in the driver package is as follows: **DBS-GaussDB-driver_*/Distributed/***Driver type*/*OS information*** (for example, **Euler2.10_X86_64**).

  The target package name is **GaussDB-Kernel**_*Database version number*_**Odbc**_*CPU architecture information*_**Distributed.tar.gz**. After decompressing this package, you will obtain the required ODBC package, named **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Odbc.tar.gz**.

  When developing applications in Linux environments, you will need unixODBC header files such as **sql.h** and **sqlext.h**, as well as the **libodbc.so** library. The header files and library can be obtained from the unixODBC-2.3.7 source code package.

- Windows OS:

  Obtain the ODBC package from the driver package. Different variants of the driver package are provided for various database kernel OSs. Select the one matching your actual database kernel OS. The file hierarchy in the driver package is as follows: **DBS-GaussDB-driver_*/Distributed/***Driver type*/*OS information*** (for example, **Euler2.10_X86_64**).

  After decompressing the two packages named with "Windows" within this file, you will obtain **GaussDB-Kernel**_*Database version number*_**Windows_X86_Odbc.tar.gz** (32-bit) and **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** (64-bit).

  In Windows environments, the header files and library required for application development are readily available within the OS, eliminating the need to obtain them manually.

5.4.2.2 Connecting to a Database

5.4.2.2.1 Configuring a Data Source on Linux

Before accessing a database through ODBC, ensure that the ODBC driver and its dependencies have been correctly installed in your environment. Application-database communication relies on a data source name (DSN), as the ODBC driver manager requires a DSN to load the appropriate driver, establish connections, and provide unified APIs for applications.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

This section explains how to configure an ODBC data source (including the requirements for editing the driver and DSN configuration files) on Linux and describes the role of these configurations in the connection process. Once configured, applications can access databases through standard ODBC APIs.

Procedure

Step 1 Install unixODBC. (The unixODBC source code package has been obtained by default.) If unixODBC is already installed, you can directly overwrite it with a new version.

For example, to install unixODBC-2.3.7, run the following commands on the client:

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no # To compile on an Arm server, append the --build=aarch64-unknown-linux-gnu parameter to configure.
make
# The installation may require root permissions.
make install
```

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- By default, it is installed in the **/usr/local** directory. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to enhance performance. However, for applications that pass invalid handles to ODBC APIs, this option will cause an abrupt failure instead of an SQL_INVALID_HANDLE error.

Step 2 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Odbc.tar.gz**. After decompression, the **lib** and **odbc** directories are present, with the **odbc** directory containing a subdirectory named **lib**. Copy all dynamic libraries from **/lib** and **/odbc/lib** to **/usr/local/lib**.

Step 3 Configure a data source.

1. Configure the ODBC driver file.

   Append the following content to the **/usr/local/etc/odbcinst.ini** file:

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/gsqlodbcw.so
   setup=/usr/local/lib/gsqlodbcw.so
   ```

   Table 5-55 describes the parameters in the **odbcinst.ini** file.

Table 5-55 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to the driver in the DSN.	[GaussMPP]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/gsqlodbcw.so

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/gsqlodbcw.so

2. Configure the data source file.

   Append the following content to the **/usr/local/etc/odbc.ini** file:

   ```
   [gaussdb]
   Driver=GaussMPP
   # Specify the IP address of the database server.
   Servername=127.0.0.1
   # Specify the database name.
   Database=db1
   # Specify the database username.
   Username=omm
   # Enter the password for the database user.
   Password=
   # Specify the listening port of the database.
   Port=8000
   Sslmode=allow
   ```

   To modify the parameters (such as server IP address, port number, database name, or authentication information) of an existing data source, you can directly edit the corresponding DSN entry in the **odbc.ini** file and update related configuration items. Once saved, the updated configurations will take effect the next time the application establishes a connection.

   If a data source is no longer used, you can delete the entire DSN configuration section from the **odbc.ini** file. Applications that depend on this DSN will then no longer be able to establish ODBC connections using it.

   Table 5-142 describes the parameters in the **odbc.ini** file.

Step 4 Configure environment variables on the client.

```
vim ~/.bashrc
```

Append the following content to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 5 Run the following command to validate the addition:

```
source ~/.bashrc
```

Step 6 Test the connection.

After installation, the generated binary files are stored in the **/usr/bin** directory.

Run the following command (where **gaussdb** is an example DSN):

```
isql -v gaussdb
```

- If the following information is displayed, the configuration is correct and the connection succeeds:

  ```
  +---------------------------------------+
  | Connected!                            |
  |                                       |
  | sql-statement                         |
  | help [tablename]                      |
  | quit                                  |
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
|                                      |
+--------------------------------------+
```

- If error information is displayed, the configuration is incorrect. Check whether the preceding configuration steps are correctly performed.
- In a cluster environment, you need to copy and configure the unixODBC file on all nodes.

NOTE

When ODBC is used to connect to a database, the kernel parameters are set as follows:

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

These parameters may cause the ODBC and gsql clients to display inconsistent data in aspects such as date data display mode and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.

In M-compatible databases, **extra_float_digits** defaults to **0**.

----End

Troubleshooting

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/gsqlodbcw.so':file not found.

  Possible causes:

  – The path configured in the **odbcinst.ini** file is incorrect.

    Run **ls** to check the path in the error information, ensuring that the **gsqlodbcw.so** file exists and you have execution permissions on it.

  – The dependency library of **gsqlodbcw.so** does not exist or is not in system environment variables.

    Run the **ldd** command to check the path in the error information. If the unixODBC library such as **libodbc.so.1** is missing, reconfigure unixODBC according to the procedure, ensure that the **lib** directory in the installation path is added to *LD_LIBRARY_PATH*. If the problem persists after reinstallation, manually copy the content in the **unixodbc/lib** directory of the database installation package to the **lib** directory in the installation path of the unixODBC. If other libraries do not exist, add the **lib** directory under the ODBC driver package to *LD_LIBRARY_PATH*. If other standard libraries are missing, install them.

- [UnixODBC]connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database address or port is configured.

    Check the **Servername** and **Port** configuration items in data sources.

  – Server listening configurations are incorrect.

    If **Servername** and **Port** are correctly configured, verify the configured NIC and port based on database server configurations in **Procedure**.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, and ensure that the database communication port is trusted.

    Check to ensure that network gatekeeper settings are proper (if any).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- [unixODBC]The password-stored method is not supported.

  Possible causes:

  The **Sslmode** configuration item is not configured in the data source.

  Set the configuration item to **allow** or a higher level. For details, see **Table 5-143**.

- Server common name "xxxx" does not match host name "xxxxx"

  Possible causes:

  When **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one.

  To solve this problem, use **verify-ca** to stop checking host names, or generate a set of server certificates containing the actual host names.

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

  Possible causes:

  The executable file (such as the **isql** of unixODBC) and the database driver (**gsqlodbcw.so**) depend on different library versions of ODBC, such as **libodbc.so.1** and **libodbc.so.2**. You can verify this problem by using the following method:

  ```
  ldd `which isql` | grep odbc
  ldd gsqlodbcw.so | grep odbc
  ```

  If the suffix digits of the outputs **libodbc.so** are different or indicate different physical disk files, this problem exists. Both **isql** and **gsqlodbcw.so** require **libodbc.so** to be loaded. If they load different physical files, two sets of function lists with the same name are generated in a visible domain (the **libodbc.so.**** function export lists of unixODBC are the same). This results in conflicts and the database driver cannot be loaded.

  Uninstall the unnecessary unixODBC, such as **libodbc.so.2**, and create a soft link with the same name and the .so.2 suffix for the remaining **libodbc.so.1** library.

- FATAL: Forbid remote connection with trust method!

  Possible causes:

  For security reasons, database CNs prevent unauthorized access from other nodes within the cluster.

  To access a CN from inside the cluster, deploy the ODBC program on the host of the CN and use 127.0.0.1 as the server address. To prevent any impact on database performance, it is advisable to deploy service systems outside the cluster.

- [unixODBC][Driver Manager]Invalid attribute value

  Possible causes:

  If this issue occurs during the use of the "SQL on other GaussDB" functionality, a possible cause is that the installed unixODBC version is not the recommended one.

  You are advised to run the **odbcinst --version** command to verify the unixODBC version in your environment.

- authentication method 10 not supported.

  This issue occurs when an open-source client is used. Possible causes include:

  The user password is stored only in SHA-256 hash format, with no plaintext password retained in the database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

To maintain compatibility with certain open-source clients, earlier database versions stored both MD5 and SHA-256 hash values. During user login, the server prioritized the authentication mode (MD5 or SHA-256) requested by the client to verify the corresponding hash value.

After being upgraded from these earlier versions to a later version, the database cannot reverse-engineer the original plaintext password from the stored SHA-256 hash value due to the one-way irreversible nature of the hash algorithm. Consequently, it cannot generate the corresponding MD5 hash value. After the upgrade, the database retains only the SHA-256 hash value, causing open-source clients that rely on MD5 authentication to fail authentication and report this error.

NOTE

The MD5 algorithm is prone to significant security vulnerabilities and is therefore no longer recommended. To address compatibility and security risks, you are advised to upgrade all clients to versions that support SHA-256 or stronger hash algorithms.

To resolve this issue, consider the following methods:

- Update the user password. For details, see "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.
- Create a user with the same permissions as the original user. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" in *Reference*. Then use this new user to connect to the database.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  Possible causes:

  The target database is either an outdated version or an open-source variant.

  Use a matching driver version to connect to the target database.

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  In the **gs_hba.conf** file of the target CN, the **gss** authentication mode is configured for the current client IP address. However, this authentication algorithm does not support client identity authentication. Change the configuration to **sha256** and try again.

- isql: error while loading shared libraries:xxx

  The current environment lacks dynamic library files. Install the required dynamic libraries.

5.4.2.2.2 Configuring a Data Source on Windows

The ODBC data source manager is preinstalled in Windows, eliminating the need for manual installation. You can proceed directly to configure a data source.

Procedure

Step 1 Replace the GaussDB client driver.

Decompress the 64-bit driver package **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** or 32-bit driver package **GaussDB-Kernel**_*Database version number*_**Windows_X86_Odbc.tar.gz**, and click **gsqlodbc.exe** to install the driver.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 2 Open the driver manager.

Use the ODBC driver manager corresponding to your ODBC version. (For example, a 64-bit ODBC driver requires the 64-bit ODBC driver manager. The following instructions assume the OS is installed on drive C. For other drives, adjust the paths accordingly.)

- If you want to use 32-bit ODBC driver manager in a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

  NOTE

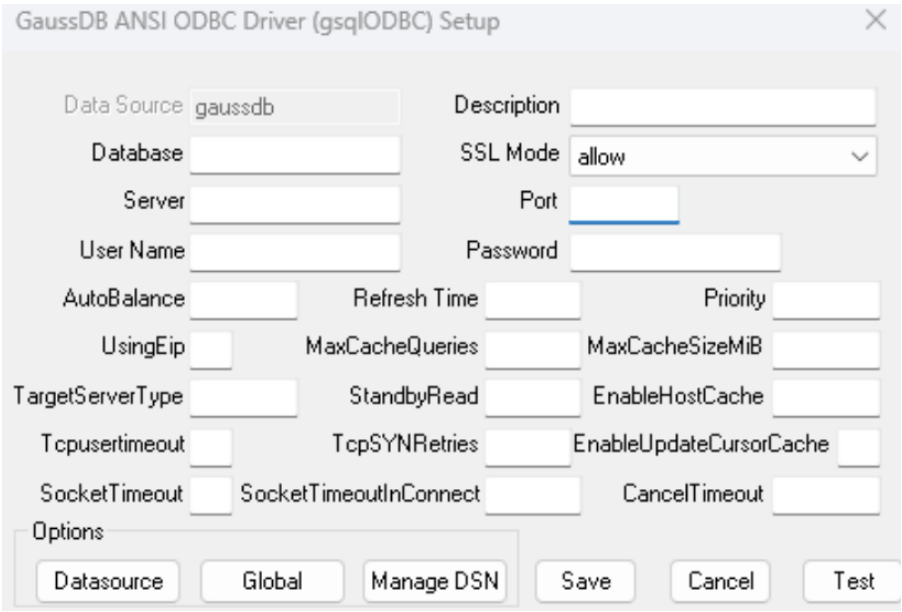
  WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64** stores the 32-bit environment running on a 64-bit system. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- On a 32-bit OS, run **C:\Windows\System32\odbcad32.exe** or choose **Computer** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to launch Driver Manager.
- On a 64-bit OS, choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to launch Driver Manager.

Step 3 Configure a data source.

On the **User DSN** tab page of Driver Manager, click **Add** and choose **GaussDB Unicode**. Then set the parameters shown in **Figure 5-14**.

Figure 5-14 ODBC Driver Setup

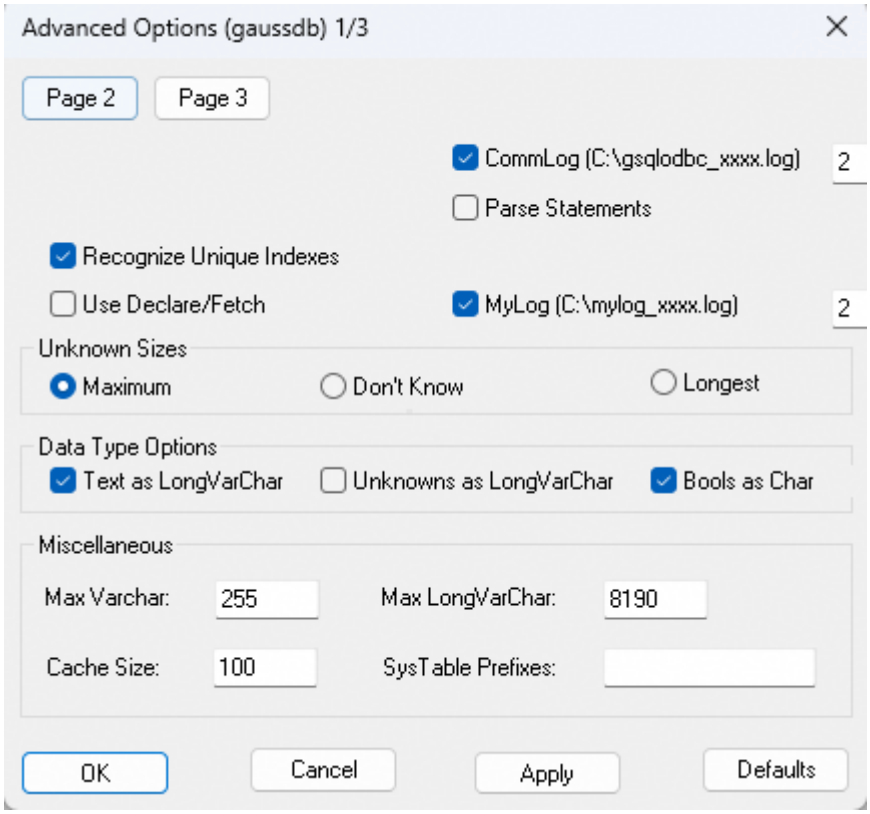


For details about the parameters, see **Table 5-142**.

You can click **Datasource** to configure whether to print logs. The dialog box shown in **Figure 5-15** is displayed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-15 Advanced Options Page1



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Enable SSL.

To use SSL, set **SSL Mode** to **require** in the window displayed in **Step 3**.

The descriptions of available SSL modes can be found in **Table 5-143**.

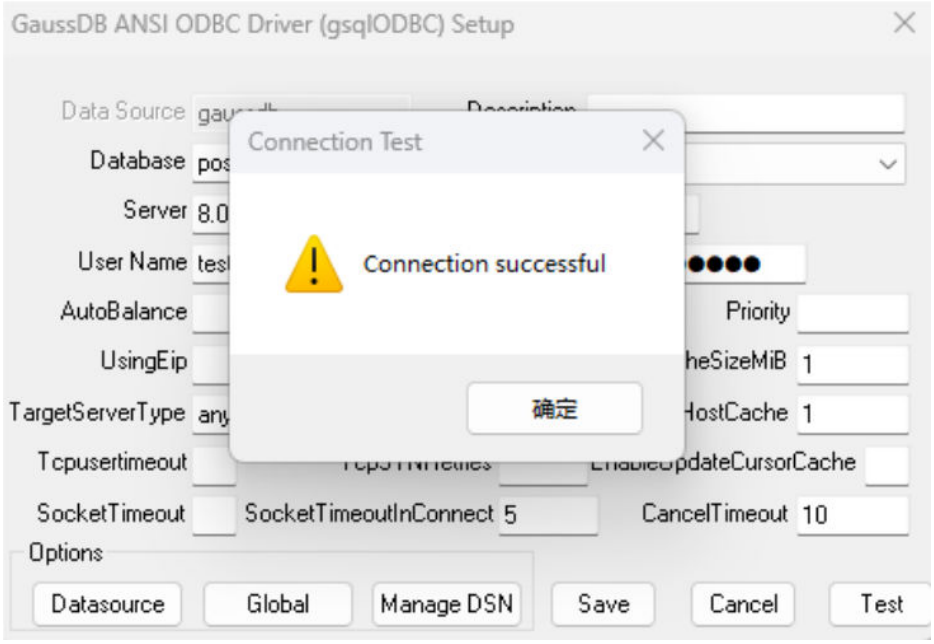
Step 5 Test the connection.

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-16 Successful configuration



- If error information is displayed, the configuration is incorrect. Check whether the preceding configurations are correct.

NOTE

When ODBC is used to connect to a database, the following parameters are set:

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

These parameters may cause the ODBC and gsql clients to display inconsistent data, for example, date data display mode and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.

In M-compatible databases, **extra_float_digits** defaults to **0**.

----End

Troubleshooting

- connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database address or port is configured.

  Check the **Server** and **Port** configuration items in data sources.

  – Server listening configurations are incorrect.

  If **Server** and **Port** are correctly configured, verify the configured NIC and port based on database server configurations in **Procedure**.

  – Firewall and network gatekeeper settings are improper.

  Check firewall settings, and ensure that the database communication port is trusted.

  Check to ensure that network gatekeeper settings are proper (if any).

- The password-stored method is not supported.

  Possible causes:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Sslmode is not set in the data source. Set it to **allow** or higher to allow SSL connections. For details about this option, see **Table 5-143**.

- authentication method 10 not supported.

  This issue occurs when an open-source client is used. Possible causes include:

  The user password is stored only in SHA-256 hash format, with no plaintext password retained in the database.

  To maintain compatibility with certain open-source clients, earlier database versions stored both MD5 and SHA-256 hash values. During user login, the server prioritized the authentication mode (MD5 or SHA-256) requested by the client to verify the corresponding hash value.

  After being upgraded from these earlier versions to a later version, the database cannot reverse-engineer the original plaintext password from the stored SHA-256 hash value due to the one-way irreversible nature of the hash algorithm. Consequently, it cannot generate the corresponding MD5 hash value. After the upgrade, the database retains only the SHA-256 hash value, causing open-source clients that rely on MD5 authentication to fail authentication and report this error.

  The database stores only SHA-256 hash values for passwords, while the open-source client supports only MD5 authentication.

  NOTE

  The MD5 algorithm is prone to significant security vulnerabilities and is therefore no longer recommended. To address compatibility and security risks, you are advised to upgrade all clients to versions that support SHA-256 or stronger hash algorithms.

  To resolve this issue, consider the following methods:

  – Update the user password. For details, see "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

  – Create a user with the same permissions as the original user. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" in *Reference*. Then use this new user to connect to the database.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  Possible causes:

  The target database is either an outdated version or an open-source variant.

  Use a matching driver version to connect to the target database.

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  In the **gs_hba.conf** file of the target CN, the **gss** authentication mode is configured for the current client IP address. However, this authentication algorithm does not support client identity authentication. Change the configuration to **sha256** and try again.

5.4.2.2.3 APIs for Connecting to a Database

After the data source is configured, applications still require a set of standard ODBC APIs to connect to databases. This connection process involves allocating environment handles and connection handles, setting connection properties, and finally establishing the connection. Related APIs are detailed in **Table 5-56**. Throughout this process, the ODBC driver manager is responsible for loading the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

driver, parsing DSN parameters, and initializing the context required for all subsequent database interactions.

Table 5-56 Related APIs

Functionality	API
Allocate a handle.	**SQLAllocHandle** is a generic function for allocating a handle. It can replace the following functions: • **SQLAllocEnv**: allocates an environment handle. • **SQLAllocConnect**: allocates a connection handle. • **SQLAllocStmt**: allocates a statement handle.
Set environment attributes.	**SQLSetEnvAttr**
Set connection properties.	**SQLSetConnectAttr**
Connect to a database.	**SQLConnect**

The following provides a brief example of developing the source program **DBtest.c** (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// DBtest.c (compile with: libodbc.so)
// Program header file and global variables. (Refer to the complete example.)

// Allocate an environment handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   printf("Error AllocHandle\n");
   exit(0);
}

// Set the version information (environment attributes).
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

// Allocate a connection handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
   exit(0);
}

// Obtain the username and password.
char *userName;
userName = getenv("EXAMPLE_USERNAME_ENV");
char *password;
password = getenv("EXAMPLE_PASSWORD_ENV");

// Set connection properties.
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

// Connect to the database. userName and password indicate the username and password for connecting to the database respectively.
// If the username and password have been set in the odbc.ini file, you can retain "". However, you are advised not to do so because the username and password will be disclosed if the permission on odbc.ini is
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
abused.
V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
  printf("Error SQLConnect %d\n",V_OD_erg);
  SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
  exit(0);
}
printf("Connected !\n");
```

5.4.2.3 Executing SQL Statements

To execute SQL statements, ODBC provides a set of dedicated APIs (detailed in **Table 5-57**). These APIs cover tasks such as SQL preprocessing, parameter binding, direct SQL execution, and statement attribute configuration. They serve as core interfaces for instructional interaction between applications and databases.

Table 5-57 Related APIs

Functionality	API
Set statement attributes.	**SQLSetStmtAttr**
Prepare an SQL statement for execution.	**SQLPrepare**
Run a prepared SQL statement.	**SQLExecute**
Bind the parameter marker of an SQL statement to a buffer.	**SQLBindParameter**
Execute an SQL statement directly.	**SQLExecDirect**

NOTE

- ODBC connects applications to the database and delivers the SQL statements sent by an application to the database. It does not parse the SQL syntax. Therefore, when confidential information (such as a plaintext password) is written into the SQL statement sent by an application, the confidential information is exposed in the driver log.
- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.
- When transaction autocommit is disabled, manual explicit commits are required. Failure to do so may result in issues. For example, a transaction is implicitly started when a SELECT statement is executed. If not manually committed, this implicit transaction cannot conclude properly and may persist as a long-running transaction, leading to a range of potential issues.

Below is a brief example (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// Allocate a statement handle.
SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Set statement attributes.
SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3, 0);

// Execute an SQL statement directly.
SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25, 'li')", SQL_NTS);

// Prepare for execution.
SQLPrepare(V_OD_hstmt, "insert into customer_t1 values(?)", SQL_NTS);

// Add parameters.
SQLBindParameter(V_OD_hstmt,1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0,
                 &value, 0, NULL);

// Run the prepared statement.
SQLExecute(V_OD_hstmt);      SQLExecDirect(V_OD_hstmt, "select c_customer_sk from customer_t1",
SQL_NTS);
```

5.4.2.4 Processing the Result Set

Processing result sets via ODBC is the action of obtaining data from the database for application processing. This includes tasks such as data retrieval, display, processing, transmission, and the implementation of service logic. **Table 5-58** describes related APIs.

Table 5-58 Related APIs

Functionality	API
Bind a buffer to a column in the result set.	**SQLBindCol**
Fetch the next row (or rows) from a result set.	**SQLFetch**
Return data in a column of a result set.	**SQLGetData**
Get the column information from a result set.	**SQLColAttribute**
Return the error message of the last operation.	**SQLGetDiagRec**

Below is a brief example (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// After the SQL statement is executed, obtain the attributes of a column in the result set.
SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE_NAME,typename,100,NULL,NULL);
printf("SQLColAttribute %s\n",typename);

// Bind the result set.
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
       (SQLLEN *)&V_OD_err);

// Use SQLFetch to obtain data from the result set.
V_OD_erg=SQLFetch(V_OD_hstmt);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Use SQLGetData to obtain and return data.
while(V_OD_erg != SQL_NO_DATA)
{
  SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
  printf("SQLGetData ----ID = %d\n",V_OD_id);
  V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");
```

5.4.2.5 Closing the Connection

After completing the desired database operations, you must properly close statements, connections, and environment handles through ODBC APIs to release resources on both driver and database sides. According to ODBC's handle management mechanism, applications must release resources in the sequence of statements → connections → environment to prevent resource leaks or driver exceptions. The ODBC provides APIs for closing database connections and releasing resources, as described in **Table 5-59**.

Table 5-59 API description

Function	API
Disconnect from a data source.	**SQLDisconnect**
Release a handle.	**SQLFreeHandle** is a generic function for releasing a handle. It can replace the following functions: • **SQLFreeEnv**: releases an environment handle. • **SQLFreeConnect**: releases a connection handle. • **SQLFreeStmt**: releases a statement handle.

Below is a brief example (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// Disconnect from the data source and release handles.
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
```

5.4.3 Typical Application Development Examples

5.4.3.1 Typical Application Scenarios and Configurations

Log Diagnosis

ODBC logs are classified into unixODBC driver manager logs and gsqlODBC driver logs. The former traces the execution status of application API calls, while the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

latter contains the diagnostic (DFX) logs recording the driver's underlying implementation and is therefore helpful for locating problems.

For unixODBC logs, configure the **odbcinst.ini** file as follows:

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
Driver64=/usr/local/lib/gsqlodbcw.so
setup=/usr/local/lib/gsqlodbcw.so
```

For gsqlODBC logs, simply add the following content to **odbc.ini**:

```
[gaussdb]
Driver=GaussMPP
# IP address of the database server.
Servername=10.10.0.13
...
# Enable debug logging for the driver, with a log level of TRACE.
Debug=5
```

NOTE

unixODBC logs are generated in the directory specified by **TraceFile**. Debug logs from the gsqlODBC driver are stored in the system's **/tmp/** directory with file names like **mylog**_*xxx*.**log**.

Load Balancing

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

- Load balancing is to randomly distribute concurrent connections to all CNs to prevent a single CN from being overloaded and improve performance.
- Set **AutoBalance** to **1** to enable load balancing.
- Set **RefreshCNListTime** to **5** as required. The default refresh interval is 10s.
- Set **Priority** to **1** as required. In this case, concurrent connections are preferentially sent to the CNs configured in the configuration file. If all the configured CNs are unavailable, the connections are distributed to the remaining CNs.

Example:

Six CNs—CN1, CN2, CN3, CN4, CN5, and CN6—are configured in the cluster environment. Four CNs—CN1, CN2, CN3, and CN4—are configured in the configuration file.

Example content of the **odbc.ini** file:

```
[gaussdb]
Driver=GaussMPP
# Specify the IP address of the database server.
Servername=10.145.130.26,10.145.130.27,10.145.130.28,10.145.130.29
# Specify the database name.
Database=db1
# Specify the database username.
Username=omm
# Enter the password for the database user.
Password=
# Specify the listening port of the database.
Port=8000
Sslmode=allow
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
AutoBalance=1
RefreshCNListTime=3
Priority=1
```

If the configuration file and cluster environment are the same as those in the example, concurrent connections are randomly and evenly distributed to CN1, CN2, CN3, and CN4. When CN1, CN2, CN3, and CN4 are all unavailable, concurrent connections are randomly and evenly sent to CN5 and CN6. If any CN among CN1, CN2, CN3, and CN4 becomes available, the connections are not sent to CN5 and CN6 but to the available CN.

Scenarios of Read-on-Standby

Configure CNs in the configuration file and set **StandbyRead=1** to enable the read-on-standby mode in a distributed system.

Example content of the **odbc.ini** file:

```
[gaussdb]
Driver=GaussMPP
# Specify the IP address of the database server.
Servername=10.145.130.26,10.145.130.27,10.145.130.28,10.145.130.29
# Specify the database name.
Database=db1
# Specify the database username.
Username=omm
# Enter the password for the database user.
Password=
# Specify the listening port of the database.
Port=8000
# Enable the read-on-standby mode.
StandbyRead=1
Sslmode=allow
# Enable load balancing mode.
AutoBalance=1
```

Connection Pool Scenario

The connection pool allows applications to reuse pre-established connections without re-establishing connections each time. Once a connection is created and put into the connection pool, the application can reuse the connection, avoiding repeated execution of the complete connection process.

The connection pool mechanism significantly enhances performance, especially for middle-layer applications that frequently establish and close connections or for applications that require network connections.

In addition to the performance advantage, the connection pool architecture enables connections in the environment to be shared by multiple components in a single process. This allows different components in the same process to utilize pooled connections without mutual interference, thereby further improving system operational efficiency and resource efficiency.

NOTE

Within the connection pool, an open connection may be reused by multiple users. Therefore, if your application scripts modify the database connection status, data status conflicts or leaks may occur. For security reasons, carefully evaluate the suitability of connection pool usage in your specific scenario.

Configuration in the Linux OS

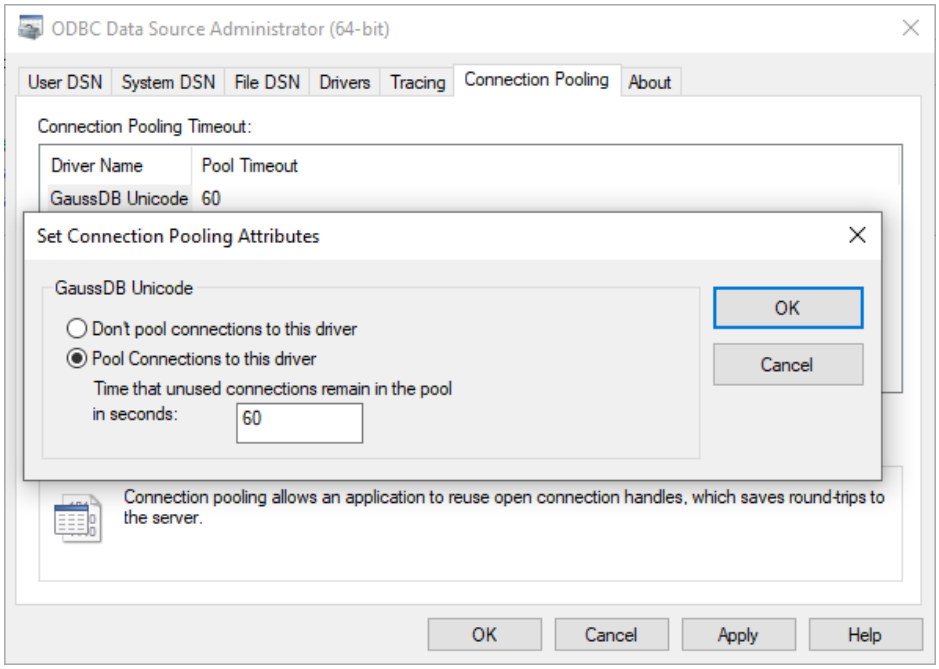
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Enable the connection pool in the **odbcinst.ini** configuration file. The reference configuration of the connection pool is as follows:

```
[ODBC]
# Enable the connection pool.
Pooling=Yes
[GaussMPP]
# Define the duration for which a connection can remain idle in the connection pool. When this period elapses, the connection will be released. The default value is 0. To enable the connection pool, set this parameter to a value greater than 0.
CPTimeout=60
# Specify the lifetime of the connection pool in the driver.
CPTimeToLive=60
[GaussMPP2]
# Disable the connection pool.
CPTimeout=0
```

Configuration in the Windows OS

On the **Connection Pooling** tab page of the driver manager, double-click the driver name **GaussDB Unicode**. Then select **Pool Connections to this driver** (default: 60s), same as the **CPTimeout** configuration for the Linux OS. Refer to the image below.



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

1. If an application requires connection pool parameters, you must call SQLSetEnvAttr to set these parameters before creating an environment handle. The environment handle must be set to **NULL** to turn **SQL_ATTR_CONNECTION_POOLING** into a process-level attribute. Currently, **SQL_ATTR_CONNECTION_POOLING** can be set to either of the following values on Windows:
   - **SQL_CP_OFF**: The connection pool feature is disabled. This is the default.
   - **SQL_CP_ONE_PER_DRIVER**: The connection pool feature is enabled. Each driver has a connection pool, and all its connections share this single pool.
2. When an application calls SQLConnect or SQLDriverConnect, a connection is retrieved from the connection pool. If the connection times out or no pooled connection matches the request, a new connection will be established. The connection pool operates transparently to the calling application.
3. When an application calls SQLDisconnect, the connection is returned to the connection pool for future reuse rather than being directly released.
4. Before SQLFreeHandle is called to release environment handles, all environment attributes successfully set by the application will remain active.
5. If an application's connection remains inactive (unused) for a defined period, it will be removed from the pool. The size of the connection pool is limited only by the memory and server.

5.4.3.2 Obtaining and Processing Data in a Database

This example illustrates how to obtain and process data in GaussDB through ODBC.

Prerequisite: The data source has been configured successfully. For details, see **Configuring a Data Source on Linux** or **Configuring a Data Source on Windows**.

NOTE

- In Windows, ODBC application code can be compiled using the Minimalist GNU for Windows (MinGW) compiler. The compilation command is as follows:
  ```
  gcc  odbctest.c -o odbctest -lodbc32
  ```
  Run the following command:
  ```
  ./odbctest.exe
  ```
- In Linux, ODBC application code can be compiled using the GNU Compiler Collection (GCC). The compilation command is as follows:
  ```
  gcc odbctest.c -o odbctest -lodbc
  ```
  Run the following command:
  ```
  ./odbctest
  ```
  If **sql.h** or API cannot be found during compilation, manually connect to the header file and dynamic library of unixODBC.
  ```
  gcc -I /home/omm/unixodbc/include -L /home/omm/unixodbc/lib odbctest.c -o odbctest -lodbc
  ```

```
// DBtest.c (compile with: libodbc.so)
// This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your local environment (and adjust the variable names as needed).
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
SQLHENV      V_OD_Env;      // Handle ODBC environment
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
SQLHSTMT      V_OD_hstmt;      // Handle statement
SQLHDBC       V_OD_hdbc;       // Handle connection
char          typename[100];
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. Allocate an environment handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }

    // 2. Set the version information (environment attributes).
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

    // 3. Allocate a connection handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }

    // Obtain the username and password.
    char *userName;
    userName = getenv("EXAMPLE_USERNAME_ENV");
    char *password;
    password = getenv("EXAMPLE_PASSWORD_ENV");

    // 4. Set connection properties.
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

    // 5. Connect to the database. userName and password indicate the username and password for
connecting to the database, respectively.
    // If the username and password have been set in the odbc.ini file, you can retain "". However, you are
advised not to do so because the username and password will be disclosed if the permission for odbc.ini is
abused.
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                          (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
       printf("Error SQLConnect %d\n",V_OD_erg);
       SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
       exit(0);
    }
    printf("Connected !\n");

    // 6. Allocate a statement handle.
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

    // 7. Set statement attributes.
    SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3,0);

    // 8. Execute SQL statements.
    SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
    SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
    SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25,'li')", SQL_NTS);

    // 9. Prepare for execution.
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);

    // 10. Bind parameters.
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                     &value,0,NULL);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    // 11. Run prepared statements.
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);

    // 12. Obtain attributes of a specific column in the result set.
    SQLColAttribute(V_OD_hstmt,1, SQL_DESC_TYPE_NAME,typename,100, NULL, NULL);
    printf("SQLColAttribute %s\n",typename);

    // 13. Bind the result set.
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);

    // 14. Obtain data in the result set by executing SQLFetch.
    V_OD_erg=SQLFetch(V_OD_hstmt);

    // 15. Obtain and return data by executing SQLGetData.
    while(V_OD_erg != SQL_NO_DATA)
    {
        SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");

    // 16. Disconnect data source connections and release handles.
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

The result is as follows:

```
Connected !
SQLColAttribute int4
SQLGetData ----ID = 25
SQLGetData ----ID = 100
Done!
```

5.4.3.3 Batch Binding

Prerequisite: The data source has been configured successfully. For details, see **Configuring a Data Source on Linux** or **Configuring a Data Source on Windows**.

```
/**************************************************************************
* Enable UseBatchProtocol in the data source and set the database parameter support_batch_bind to on.
* The CHECK_ERROR command is used to check and print error information.
* This example is used to interactively obtain the DSN, volume of data to be processed, and volume of
ignored data from users, and insert required data into the test_odbc_batch_insert table.
**************************************************************************/
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // Allocate a statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }

    // Prepare a statement.
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }

    // Execute the statement.
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }

    // Release the handle.
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}
int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    long     int      batchCount = 1000; // Amount of data that is bound in batches
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0;           // Amount of data that is not imported to the database among the data
that is bound in batches
    int      i = 0;
    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];
    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);

    // Allocate an environment handle.
    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

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

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Allocate connections.
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

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

// Set login timeout.
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

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

// Set the automatic commit.
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                            (SQLPOINTER)(1), 0);

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

// Connect to the database.
sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                     (SQLCHAR*) NULL, 0, NULL, 0);

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

// Initialize the table information.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");
// The following code constructs the data to be inserted based on the data volume entered by users:
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    SQLCHAR      *sql = NULL;
    SQLINTEGER   *ids  = NULL;
    SQLCHAR      *cols = NULL;
    SQLLEN       *bufLenIds = NULL;
    SQLLEN       *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
    SQLUSMALLINT *statusptr = NULL;
    SQLULEN      process = 0;

    // Data is constructed by column. Each column is stored continuously.
    ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
    cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);

   // Data size in each row for a column
    bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
    bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);

    // Specifies whether this row needs to be processed. The value is SQL_PARAM_IGNORE or
SQL_PARAM_PROCEED.
    operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
    memset(operptr, 0, sizeof(operptr[0]) * batchCount);

   // Processing result of the row
   // Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a
unit. Either all data is inserted successfully or all data fails to be inserted.
    statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);
    if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
    {
        fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
        goto exit;
    }
    for (i = 0; i < batchCount; i++)
    {
        ids[i] = i;
        sprintf(cols + 50 * i, "column test value %d", i);
        bufLenIds[i] = sizeof(ids[i]);
        bufLenCols[i] = strlen(cols + 50 * i);
        operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
    }

    // Allocate a statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

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

    // Prepare a statement.
    sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
    retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmtinesrt) failed");
        goto exit;
        retcode = SQLRowCount(hstmtinesrt, &rowsCount);

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

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

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

        // Check the number of returned rows.
        if (rowsCount != process)
        {
            sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);

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

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        for (i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

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

```
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

                if (!SQL_SUCCEEDED(retcode)) {
                    printf("SQLExecute failed");
                    goto exit;
                }
            }
        }
        retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
        sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLFreeHandle failed");
            goto exit;
        }
    }
  }
exit:
  (void) printf ("\nComplete.\n");

  // Close the connection.
  if (hdbc != SQL_NULL_HDBC) {
    SQLDisconnect(hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
  }
  // Release an environment handle.
  if (henv != SQL_NULL_HENV)
    SQLFreeHandle(SQL_HANDLE_ENV, henv);
  return 0;
}
```

The result is as follows:

```
Complete.
```

Part of database query result is as follows:

```
 id  |          col
-----+-----------------------
   0 | column test value 0
   1 | column test value 1
   2 | column test value 2
   3 | column test value 3
   4 | column test value 4
   5 | column test value 5
   6 | column test value 6
   7 | column test value 7
   8 | column test value 8
...
```

5.4.3.4 High-Performance Binding Type

If a large volume of data needs to be inserted into a database, consider the following recommendations and configurations:

- To activate batch binding, set **UseBatchProtocol** to **1** in the configuration file **odbc.ini**, and set the database GUC parameter **support_batch_bind** to **on**.
- Keep ODBC's application binding type consistent with that in the database.
- Ensure that the client's character set is consistent with the database's character set.
- Switch transactions to the manual commit mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Configuration file **odbc.ini**:

```
[gaussdb]
Driver=GaussMPP
# IP address of the database server.
Servername=10.10.0.13
...
# Enabled by default
UseBatchProtocol=1
# Set the client-side encoding, which should be the same as the server-side encoding.
ConnSettings=set client_encoding=UTF8
```

Example binding type:

Prerequisite: The data source has been configured successfully. For details, see **Configuring a Data Source on Linux** or **Configuring a Data Source on Windows**.

```
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN  128
SQLHANDLE   h_env = NULL;
SQLHANDLE   h_conn = NULL;
SQLHANDLE   h_stmt = NULL;
void print_error()
{
   SQLCHAR     Sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER  NativeError;
   SQLCHAR     MessageText[MESSAGE_BUFFER_LEN];
   SQLSMALLINT TextLength;
   SQLRETURN   ret = SQL_ERROR;

   ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
   if ( SQL_SUCCESS == ret)
   {
      printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
      return;
   }

   ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
   if ( SQL_SUCCESS == ret)
   {
      printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
      return;
   }

   ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
   if ( SQL_SUCCESS == ret)
   {
      printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
      return;
   }

   return;
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS(func) \
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS_WITH_INFO. */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the values are the same. */
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
}

/* Expect the character strings are the same. */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}


/* prepare + execute SQL statement */
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}
/* execute + commit handle */
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    /* Manual commit */
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER    ret;

    /* Allocate an environment handle. */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }

    /* Set the version number before connection. */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
0))
    {
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a connection handle. */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Establish a connection. */
    ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                            (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect  failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a statement handle. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
    /* Release a statement handle. */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* Release a connection handle. */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* Release an environment handle. */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
}

int main()
{
    /* begin test */
    if (begin_unit_test() != SQL_SUCCESS)
    {
        printf("\n begin_test_unit failed.");
        return SQL_ERROR;
    }
    /* The handle configuration is the same as that in the preceding case. */
    int         i = 0;
    SQLCHAR*    sql_drop   = "drop table if exists test_bindnumber_001";
    SQLCHAR*    sql_create = "create table test_bindnumber_001("
                             "f4 number, f5 number(10, 2)"
                             ")";
    SQLCHAR*    sql_insert = "insert into test_bindnumber_001 values(?, ?)";
    SQLCHAR*    sql_select = "select * from test_bindnumber_001";
    SQLLEN      RowCount;
    SQL_NUMERIC_STRUCT st_number;
    SQLCHAR     getValue[2][MESSAGE_BUFFER_LEN];

    /* Step 1. Create a table. */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

    /* Step 2.1 Bind parameters using the SQL_NUMERIC_STRUCT structure. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

    /* First line: 1234.5678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 4;
    st_number.sign = 1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

    /* Disable the automatic commit function. */
    SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Second line: 12345678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 0;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Third line: 12345678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 0;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);


    /* Step 2.2 Bind parameters by using the SQL_C_CHAR character string in the fourth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLCHAR*    szNumber = "1234.5678";
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.3 Bind parameters by using SQL_C_FLOAT in the fifth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLREAL     fNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.4 Bind parameters by using SQL_C_DOUBLE in the sixth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLDOUBLE   dNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLBIGINT   bNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLBIGINT   bNumber2 = 12345;

    /* Step 2.5 Bind parameters by using SQL_C_SBIGINT in the seventh line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.6 Bind parameters by using SQL_C_UBIGINT in the eighth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLLEN  lNumber2 = 12345;

    /* Step 2.7 Bind parameters by using SQL_C_LONG in the ninth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.8 Bind parameters by using SQL_C_ULONG in the 10th line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLSMALLINT sNumber = 0xFFFF;

    /* Step 2.9 Bind parameters by using SQL_C_SHORT in the 11th line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.10 Bind parameters by using SQL_C_USHORT in the 12th line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLCHAR cNumber = 0xFF;

    /* Step 2.11 Bind parameters by using SQL_C_TINYINT in the 13th line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.12 Bind parameters by using SQL_C_UTINYINT in the 14th line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Use the character string type to unify the expectation. */
    SQLCHAR*    expectValue[14][2] = {{"1234.5678",            "1234.57"},
                                      {"12345678",             "12345678"},
                                      {"0",                    "0"},
                                      {"1234.5678",            "1234.57"},
                                      {"1234.5677",            "1234.57"},
                                      {"1234.5678",            "1234.57"},
                                      {"-1",                   "12345"},
                                      {"18446744073709551615",  "12345"},
                                      {"-1",                   "12345"},
                                      {"4294967295",           "12345"},
                                      {"-1",                   "-1"},
                                      {"65535",                "65535"},
                                      {"-1",                   "-1"},
                                      {"255",                  "255"},
                                      };

    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
    while ( SQL_NO_DATA != SQLFetch(h_stmt))
    {
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));
        i++;
    }
    printf("\nComplete!\n");
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(i, RowCount);
    SQLCloseCursor(h_stmt);
    /* Final step. Drop the table and restore the environment. */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(commit_exec());
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    end_unit_test();
}
```

The result is as follows:

```
Complete!
```

5.4.3.5 Enabling a Connection Pool

Prerequisite: The data source has been configured successfully. For details, see **Configuring a Data Source on Linux** or **Configuring a Data Source on Windows**.

Enabling a connection pool in a Linux or Windows OS can significantly improve the performance of middle-layer applications that need to establish and close connections frequently. For details about the reference configuration, see **Connection Pool Scenario**. The sample code is as follows:

```
#ifdef WIN32
#include <windows.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>
#include <pthread.h>
#include <sqltypes.h>
#include <time.h>
SQLHENV env;
SQLHDBC conn;
struct timeval start, end;

#define CONN_COUNT 15000
#define CHECK_ERROR(retcode, str, handle, htype)                    \
    ({                                                              \
        if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO) { \
            fprintf(stderr, "FAILED:\t");                           \
            extract_error(str, handle, htype);                      \
            exit(-1);                                               \
        } else {                                                    \
            printf("OK:\t%s\n", str);                               \
        }                                                           \
    })

void print_diag(char *msg, SQLSMALLINT htype, SQLHANDLE handle)
{
    char sqlstate[32];
    char message[1000];
    SQLINTEGER nativeerror;
    SQLSMALLINT textlen;
    SQLRETURN ret;

    if (msg) {
        printf("%s\n", msg);
    }

    ret = SQLGetDiagRec(htype, handle, 1, sqlstate, &nativeerror, message, 256, &textlen);

    if (ret != SQL_ERROR) {
        printf("%s=%s\n", (CHAR *)sqlstate, (CHAR *)message);
    }
}

void extract_error(char *fn, SQLHANDLE handle, SQLSMALLINT type)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
{
    SQLINTEGER i = 0;
    SQLINTEGER NativeError;
    SQLCHAR SQLState[7];
    SQLCHAR MessageText[256];
    SQLSMALLINT TextLength;
    SQLRETURN ret;

    fprintf(stderr, "The driver reported the following error %s\n", fn);
    if (NULL == handle)
        return;
    do {
        ret = SQLGetDiagRec(type, handle, ++i, SQLState, &NativeError, MessageText, sizeof(MessageText),
&TextLength);
        if (SQL_SUCCEEDED(ret)) {
            printf("[SQLState:%s]:[%ldth error]:[NativeError:%ld]: %s\n",
                SQLState,
                (long)i,
                (long)NativeError,
                MessageText);
        }
    } while (ret == SQL_SUCCESS);
}

void InitializeEnvironment()
{

    /* Enable the connection pool. Configure connection pool parameters in the Windows OS before
allocating an environment handle. */
    SQLSetEnvAttr(env, SQL_ATTR_CONNECTION_POOLING, (SQLINTEGER *)SQL_CP_ONE_PER_DRIVER, 0);

    /* Close the connection pool in the Windows OS. */
    // SQLSetEnvAttr(env, SQL_ATTR_CONNECTION_POOLING, (SQLINTEGER*)SQL_CP_OFF, 0);

    // Allocate an environment handle.
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    // Configure the ODBC version.
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
    // Configure the timeout interval for establishing a connection.
    SQLSetConnectAttr(conn, SQL_LOGIN_TIMEOUT, (SQLPOINTER *)5, 0);
}

void test_connect()
{
    SQLRETURN ret;
    SQLCHAR str[1024];
    SQLSMALLINT strl;
    SQLCHAR dsn[1024];
    SQLUINTEGER uIntVal;

    SQLAllocHandle(SQL_HANDLE_DBC, env, &conn);

    /* Adjust the connection string based on the scenario. */
    char *config = "Driver=GaussMPP;DSN=gaussdb;";
    ret = SQLSetConnectAttr(conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)(1), 0);

    ret = SQLDriverConnect(conn, 0, (SQLCHAR *)config, SQL_NTS, (SQLCHAR *)NULL, SQL_NTS, 0,
SQL_DRIVER_NOPROMPT);

    if (SQL_SUCCEEDED(ret)) {
        // printf("Connected\n");
    } else {
        print_diag("SQLDriverConnect failed.", SQL_HANDLE_DBC, conn);
        SQLFreeHandle(SQL_HANDLE_DBC, conn);
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        exit(1);
    }

    /* Put the connection into the connection pool to reuse the connection. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    if (conn != SQL_NULL_HDBC) {
        SQLDisconnect(conn);
        SQLFreeHandle(SQL_HANDLE_DBC, conn);
        conn = SQL_NULL_HDBC;
    }
}

int main()
{
    int count = 0;
    int timeuser;
    gettimeofday(&start, NULL);
    InitializeEnvironment();

    for (int i = 0; i < CONN_COUNT; i++) {
        test_connect();
        count++;
    }

    // Release an environment handle.
    SQLFreeHandle(SQL_HANDLE_ENV, env);
    printf("Connection count: %d\n", count);
    gettimeofday(&end, NULL);
    timeuser = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("Connection time: %.3f s \n", (double)timeuser / 1000000);
    return 0;
}
```

The result varies with the environment. When the connection pool is enabled, the running result of this example is as follows:

```
Connection count: 15000
Connection time: 14.175 s
```

When the connection pool is disabled, the running result of this example is as follows:

```
Connection count: 15000
Connection time: 691.768 s
```

NOTE

The application code implementation remains consistent across Windows and Linux environments. However, connection strings must be configured according to specific scenarios. During cross-platform deployment, pay attention to the adaptability of database connection parameters.

5.4.4 ODBC API Reference

ODBC provides a set of APIs for users. This section covers only the most commonly used APIs. To explore other APIs, refer to "ODBC Programmer's Reference" at the MSDN website (https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx).

5.4.4.1 SQLAllocEnv

In ODBC 3.x, SQLAllocEnv (an ODBC 2.x function) is replaced by SQLAllocHandle.

For details, see **SQLAllocHandle**.

5.4.4.2 SQLAllocConnect

In ODBC 3.x, SQLAllocConnect (an ODBC 2.x function) is replaced by SQLAllocHandle.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details, see **SQLAllocHandle**.

5.4.4.3 SQLAllocHandle

Description

Allocates environment, connection, statement, or descriptor handles. This function replaces the deprecated ODBC 2.x functions SQLAllocEnv, SQLAllocConnect, and SQLAllocStmt.

Prototype

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

Parameters

Table 5-60 SQLAllocHandle parameters

Keyword	Description
HandleType	Type of handle to be allocated by SQLAllocHandle. The value must be one of the following: • **SQL_HANDLE_ENV** (environment handle) • **SQL_HANDLE_DBC** (connection handle) • **SQL_HANDLE_STMT** (statement handle) • **SQL_HANDLE_DESC** (descriptor handle) The handle allocation sequence is: environment handle > connection handle > statement handle. A later allocated handle depends on a previously allocated handle.
InputHandle	Existing handle to use as a context for the new handle being allocated. The **InputHandle** parameter specifies the parent handle of the handle to be created to establish the hierarchical relationship between handles. Handles of different types have different hierarchical relationships. The **InputHandle** parameter is used to specify the relationship. • If **HandleType** is **SQL_HANDLE_ENV**, the value is **SQL_NULL_HANDLE**, indicating that there is no parent handle. • If **HandleType** is **SQL_HANDLE_DBC**, this must be an environment handle, indicating that the connection handle is created in this environment. • If **HandleType** is set to **SQL_HANDLE_STMT** or **SQL_HANDLE_DESC**, this parameter value must be a connection handle.
OutputHandlePtr	**Output parameter**: **OutputHandlePtr** is a pointer that points to the SQLHANDLE type and is used to return a new handle allocated.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLAllocHandle returns **SQL_ERROR** when it is used to allocate a non-environment handle, it sets **OutputHandlePtr** to **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call the **SQLGetDiagRec** function, with **HandleType** and **Handle** set to the value of **InputHandle**, to obtain an **SQLSTATE** value. This **SQLSTATE** value provides detailed function call information.

Examples

Refer to **Typical Application Development Examples**.

5.4.4.4 SQLAllocStmt

In ODBC 3.x, SQLAllocStmt (an ODBC 2.x function) is replaced by SQLAllocHandle.

For details, see **SQLAllocHandle**.

5.4.4.5 SQLBindCol

Description

Binds application data buffers to columns in a result set.

Prototype

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-61 SQLBindCol parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. The column number starts with 0 and increases in ascending order. Column 0 is the bookmark column. If no bookmark column is set, column numbers start with 1.
TargetType	C data type in the buffer.
TargetValuePtr	**Output parameter**: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If this parameter is a null pointer, **StrLen_or_IndPtr** is a valid value.
BufferLength	Length of the buffer pointed to by **TargetValuePtr**, in bytes.
StrLen_or_IndPtr	**Output parameter**: pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

5.4.4.6 SQLBindParameter

Description

Binds parameter markers in an SQL statement to a buffer.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Prototype

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

Parameters

Table 5-62 SQLBindParameter parameters

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting with 1 and increasing in ascending order.
InputOutputType	Input/output type of the parameter. The value can be **SQL_PARAM_INPUT**, **SQL_PARAM_OUTPUT**, or **SQL_PARAM_INPUT_OUTPUT**.
ValueType	C data type of the parameter. The value can be **SQL_C_CHAR**, **SQL_C_LONG**, or **SQL_C_DOUBLE**.
ParameterType	SQL data type of the parameter. The value can be **SQL_CHAR**, **SQL_INTEGER**, or **SQL_DOUBLE**.
ColumnSize	Size of the column or expression of the corresponding parameter marker.
DecimalDigits	Decimal digit of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the storage parameter buffer.
BufferLength	Length of the buffer pointed to by **ParameterValuePtr**, in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

When binding character data that includes the '\0' character and setting the libpq parameter **supportZeroCharacter** to **1**, ensure that the length of the input string (**StrLen_or_IndPtr**) is equal to the actual length of the string with the '\0' character. If the input string is longer than the string with the '\0' character, out-of-bounds memory access may occur, resulting in a coredump on the libpq side. If the input string is shorter than the string with the '\0' character, the latter will be truncated based on the length of the former.

When **supportZeroCharacter** is set to **1**, also ensure that the character set of the client is the same as that of the database.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

5.4.4.7 SQLColAttribute

Description

Returns the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttribute(SQLHSTMT        StatementHandle,
                          SQLUSMALLINT    ColumnNumber,
                          SQLUSMALLINT    FieldIdentifier,
                          SQLPOINTER      CharacterAttributePtr,
                          SQLSMALLINT     BufferLength,
                          SQLSMALLINT     *StringLengthPtr,
                          SQLLEN          *NumericAttributePtr);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-63 SQLColAttribute parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting with 1 and increasing in ascending order.
FieldIdentifier	Field identifier of **ColumnNumber** in IRD.
CharacterAttributePtr	**Output parameter**: pointer to the buffer that returns the **FieldIdentifier** value.
BufferLength	• Indicates the length of the buffer if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to a string or a binary buffer. • Ignore this parameter if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to an integer.
StringLengthPtr	**Output parameter**: pointer to the buffer storing the total number of bytes in the character-type data of ***CharacterAttributePtr**. Ignore **BufferLength** if the data is not the character type.
NumericAttributePtr	**Output parameter**: pointer to an integer buffer in which the value of **FieldIdentifier** in the **ColumnNumber** row of the IRD is returned.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.8 SQLConnect

Description

Establishes a connection between a driver and a data source. After the connection is established, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.

Prototype

```
SQLRETURN  SQLConnect(SQLHDBC        ConnectionHandle,
                      SQLCHAR        *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR        *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR        *Authentication,
                      SQLSMALLINT    NameLength3);
```

Parameters

Table 5-64 SQLConnect parameters

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

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLConnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call the **SQLGetDiagRec** function, with **HandleType** and **Handle** set to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQL_HANDLE_DBC and **ConnectionHandle**, respectively, to obtain an **SQLSTATE** value. This **SQLSTATE** value provides detailed function call information.

Examples

Refer to **Typical Application Development Examples**.

5.4.4.9 SQLDisconnect

Description

Closes the connection associated with a database connection handle.

Prototype

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

Parameters

Table 5-65 SQLDisconnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.10 SQLExecDirect

Description

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

Prototype

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

Parameters

Table 5-66 SQLExecDirect parameters

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

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.11 SQLExecute

Description

If a statement contains a parameter marker, the SQLExecute function uses the current value of the parameter marker to execute a prepared SQL statement.

Prototype

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

Parameters

Table 5-67 SQLExecute parameters

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.12 SQLFetch

Description

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

Prototype

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

Parameters

Table 5-68 SQLFetch parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

5.4.4.13 SQLFreeStmt

In ODBC 3.x, SQLFreeStmt (an ODBC 2.x function) is replaced by SQLFreeHandle.

For details, see **SQLFreeHandle**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.14 SQLFreeConnect

In ODBC 3.x, SQLFreeConnect (an ODBC 2.x function) is replaced by SQLFreeHandle.

For details, see **SQLFreeHandle**.

5.4.4.15 SQLFreeHandle

Description

Releases resources associated with a specific environment, connection, or statement handle. It replaces the ODBC 2.x functions: SQLFreeEnv, SQLFreeConnect, and SQLFreeStmt.

Prototype

```
SQLRETURN SQLFreeHandle(SQLSMALLINT   HandleType,
                        SQLHANDLE     Handle);
```

Parameters

Table 5-69 SQLFreeHandle parameters

Keyword	Description
HandleType	Type of handle to be freed by SQLFreeHandle. The value must be one of the following: • **SQL_HANDLE_ENV** • **SQL_HANDLE_DBC** • **SQL_HANDLE_STMT** • **SQL_HANDLE_DESC** If none of them is the value of **HandleType**, SQLFreeHandle returns **SQL_INVALID_HANDLE**.
Handle	Name of the handle to be freed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Typical Application Development Examples**.

5.4.4.16 SQLFreeEnv

In ODBC 3.x, SQLFreeEnv (an ODBC 2.x function) is replaced by SQLFreeHandle.

For details, see **SQLFreeHandle**.

5.4.4.17 SQLPrepare

Description

Prepares an SQL statement to be executed.

Note that ODBC does not support the kernel reuse plan when sending prepared statements. As a result, a new plan needs to be generated for each execution, causing high CPU usage. If services have requirements on plan reuse, you are advised to use the JDBC client.

Prototype

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

Parameters

Table 5-70 SQLPrepare parameters

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

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

5.4.4.18 SQLGetData

Description

SQLGetData is used to retrieve data for a single column in the result set. It can be called multiple times to partially retrieve variable-length data.

Prototype

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

Parameters

Table 5-71 SQLGetData parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
Col_or_Param_Num	Column number for which the data retrieval is requested. The column number starts with 1 and increases in ascending order. The number of the bookmark column is 0.
TargetType	C data type in the TargetValuePtr buffer. If **TargetType** is **SQL_ARD_TYPE**, the driver uses the data type of the **SQL_DESC_CONCISE_TYPE** column in ARD. If it is **SQL_C_DEFAULT**, the driver selects a default data type according to the source SQL data type.
TargetValuePtr	**Output parameter**: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by **TargetValuePtr**.
StrLen_or_IndPtr	**Output parameter**: pointer to the buffer where the length or identifier value is returned.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

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

Examples

Refer to **Typical Application Development Examples**.

5.4.4.19 SQLGetDiagRec

Description

Returns the current values of multiple fields in a diagnostic record that contains error, warning, and status information.

Prototype

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType,
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR        *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR        *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-72 SQLGetDiagRec parameters

Keyword	Description
HandleType	Identifier that describes the type of handle required for diagnostics. The value must be one of the following: • **SQL_HANDLE_ENV** • **SQL_HANDLE_DBC** • **SQL_HANDLE_STMT** • **SQL_HANDLE_DESC**
Handle	Handle for the diagnostic data structure. Its type is indicated by **HandleType**. If **HandleType** is set to **SQL_HANDLE_ENV**, **Handle** may indicate a shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. Status records are numbered from 1.
SQLState	**Output parameter**: pointer to a buffer that saves the 5-character **SQLSTATE** code pertaining to **RecNumber**.
NativeErrorPtr	**Output parameter**: pointer to a buffer that saves the native error code.
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of **MessageText**.
TextLengthPtr	**Output parameter**: pointer to the buffer, the total number of bytes in the returned **MessageText**. If the number of bytes available to return is greater than **BufferLength**, then the diagnostics information text in **MessageText** is truncated to **BufferLength** minus the length of the null termination character.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

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

- **SQL_SUCCESS** indicates that the function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO** indicates that the **MessageText** buffer is too small to hold the requested diagnostic information. No diagnostic records are generated.
- **SQL_INVALID_HANDLE** indicates that the handle indicated by **HandleType** and **Handle** is an invalid handle.
- **SQL_ERROR** indicates that the value of **RecNumber** is less than or equal to **0** or the value of **BufferLength** is less than **0**.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, you can call SQLGetDiagRec to obtain the **SQLSTATE** value. **Table 2 SQLSTATE values** lists the possible **SQLSTATE** values.

Table 5-73 SQLSTATE values

SQLSTATE	Error	Description
HY000	General error.	An error occurs when there is no specific SQLSTATE.
HY001	Memory allocation error.	The driver is unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled.	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error.	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error.	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timeout.	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver.	The called function is not supported by the StatementHandle driver.

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.20 SQLSetConnectAttr

Description

Sets connection attributes.

Prototype

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

Parameters

Table 5-74 SQLSetConnectAttr parameters

Keyword	Description
ConnectionHandle	Connection handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the **Attribute** value. Depending on the **Attribute** value, **ValuePtr** can be a 32-bit unsigned integer, null-terminated string, binary buffer, or driver-defined value. If defined by the driver, **ValuePtr** may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLSetConnectAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call the **SQLGetDiagRec** function, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain an **SQLSTATE** value. This **SQLSTATE** value provides detailed function call information.

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.4.4.21 SQLSetEnvAttr

Description

Sets environment attributes.

Prototype

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

Parameters

Table 5-75 SQLSetEnvAttr parameters

Keyword	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. The value must be one of the following: • **SQL_ATTR_ODBC_VERSION**: ODBC version. • **SQL_CONNECTION_POOLING**: connection pool attribute. • **SQL_OUTPUT_NTS**: string type returned by the driver.
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit integer value or a null-terminated string.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

If SQLSetEnvAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call the **SQLGetDiagRec** function, with **HandleType** and **Handle** set to **SQL_HANDLE_ENV** and **EnvironmentHandle**, respectively, to obtain an **SQLSTATE** value. This **SQLSTATE** value provides detailed function call information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Typical Application Development Examples**.

5.4.4.22 SQLSetStmtAttr

Description

Sets attributes related to a statement.

Prototype

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

Parameters

Table 5-76 SQLSetStmtAttr parameters

Keyword	Description
StatementHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the **Attribute** value. Depending on the **Attribute** value, **ValuePtr** can be a 32-bit unsigned integer, null-terminated string, binary buffer, or driver-defined value. If defined by the driver, **ValuePtr** may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles are called. This value may also be returned by other APIs.

Precautions

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Typical Application Development Examples**.

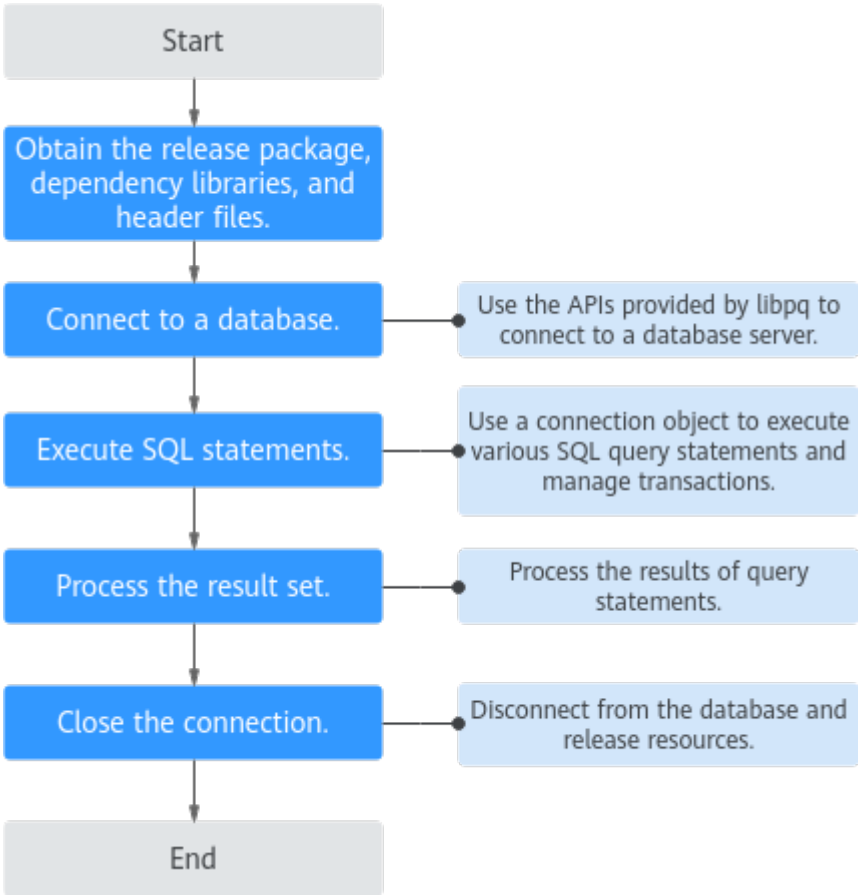
5.5 Development Based on libpq

5.5.1 Development Process

Figure 5-17 shows the development process based on libpq.

This process involves five steps: obtain the release package, dependency libraries, and header files, connect to a database, execute SQL statements, process the result set, and close the connection.

Figure 5-17 Development process based on libpq



5.5.2 Development Procedure

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.2.1 Obtaining the Release Package, Dependency Libraries, and Header Files

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 5-77**.

Table 5-77 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

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

- Verifying the software package integrity on a Linux server:
  a. Upload the software package and verification package to the same directory on the VM.
  b. Run the following command to verify the integrity of the software package:
     ```
     F=$(echo DBS-GaussDB-driver*.tar.gz);echo "$(awk '{print $1}' "$F.sha256")  $F" | sha256sum -c
     -
     ```
     If **OK** is displayed in the command output, the verification is successful. Example:
     ```
     DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz: OK
     ```
- Verifying the software package integrity on a Windows server:
  a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.
  b. Run the following command to obtain the hash value of the driver package:

     certutil -hashfile *{local_directory_of_the_driver_package}**{driver_package_name}* **sha256**

     - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
     - Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz**.

     Example: **certutil -hashfile** *C:|Users**DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz* **sha256**
  c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-77**.
     - If they are consistent, the verification is successful.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtain the dependency libraries and header files of libpq from the driver package. Different variants of the driver package are provided for various database kernel OSs. Select the one matching your actual database kernel OS. The file hierarchy in the driver package is as follows: **/Distributed/***Driver type***/***OS information* (for example, **Euler2.10_X86_64**). The file contains two packages: **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Libpq.tar.gz** and **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Libpq_Static.tar.gz**.

The Libpq_Static driver package and libpq driver package are functionally equivalent, allowing users to select the appropriate **libpq.so** based on their specific scenarios. The differences are outlined below:

- The **libpq.so** in the libpq driver package dynamically links to OpenSSL libraries (**libssl.so** and **libcrypto.so**).
- The **libpq.so** file in the Libpq_Static driver package statically links to OpenSSL libraries to resolve potential conflicts caused by version mismatches with the OS's OpenSSL in certain customer environments.

After decompressing the preceding packages, you will obtain the **include** and **lib** directories. The **include** directory contains the required header files, and the **lib** directory contains the required libpq library files. Applications using libpq must include the header file **libpq-fe.h** and connect to the libpq library located in the **lib** directory.

NOTE

In addition to **libpq-fe.h**, the **include** directory contains three header files that **libpq-fe.h** depends on by default: **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h**.

Development based on libpq must be performed on Linux OS. The following compilation environment is recommended:

- OS: The OS version should preferably align with the database kernel version.
- Compiler: GCC 7.3 or later is recommended.
- Build tools: Make and CMake (CMake 3.10 or later is recommended).

Prior to compilation, ensure that the preceding tools have been correctly installed and can be successfully called from the command line.

5.5.2.2 Connecting to a Database

Connecting to a database is the first step in developing an application using libpq. In this case, you can use the **PQconnectdb** or **PQsetdbLogin** function to connect to the database server.

The connection string **conninfo** supports the following formats:

1. Required connection parameters are directly written in *[name=value][...]* format, with multiple parameters separated by spaces. For example:
   ```
   "dbname=%s port=%s host=%s application_name=%s connect_timeout=%d sslmode=%s user=%s password=%s connectMode=%s tenantName=%s tenantPassword=%s"
   ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. The connection string is written in the format of **postgresql://***[userspec@][hostspec][/dbname][?paramspec]*, where *userspec = user[:password]*, *hostspec = [host][:port][,...]*, and *paramspec = name=value[&...]*. For example:
   ```
   "postgresql://HOST?dbname=postgres&port=PORT&user=USER&password=******"
   ```

The following provides a brief example of developing the source program **testlibpq.c** (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example):

```
/*
 * Note: The source program testlibpq.c covers basic and common application scenarios of libpq.
 * Use the PQconnectdb, PQexec, PQntuples, and PQfinish APIs of libpq to establish a database connection, execute SQL statements, obtain results, and release resources, respectively.
 * Header file. For details about user-defined functions, see the complete example.
 */

/* The values of variables such as user and passwd are read from environment variables or configuration files. When using environment variables, users must configure them manually. In other cases, users can assign values directly as character strings. */
const char conninfo[1024];
PGconn     *conn;
PGresult   *res;
int         nFields;
int         i,j;
char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
char        *port = getenv("EXAMPLE_PORT_ENV");
char        *host = getenv("EXAMPLE_HOST_ENV");
char        *username = getenv("EXAMPLE_USERNAME_ENV");
char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

/*
char        *tenant_conn_mode = getenv("EXAMPLE_TENANT_MODE_ENV");
char        *tenant_username = getenv("EXAMPLE_TENANT_USERNAME_ENV");
char        *tenant_passwd = getenv("EXAMPLE_TENANT_PASSWD_ENV");
*/

/*
 * If users provide a value for the conninfo string in the command, that value will be used.
 * Otherwise, environment variables or all other connection parameters
 * use the default values.
 */
if (argc > 1)
    strcpy(conninfo, argv[1]);
else
    sprintf(conninfo,
        "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s password=%s",
        dbname, port, host, username, passwd);


/* Connect to the database. */
conn = PQconnectdb(conninfo);

/* Verify that the backend connection has been successfully established. */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}
```

Precautions:

- It is not advisable to hardcode sensitive information (such as passwords). Instead, obtain such information from environment variables, configuration files, or key management systems.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- After a connection is established, check its status using PQstatus and handle any errors to enhance program robustness.

5.5.2.3 Executing SQL Statements

Through connection objects, you can use **PQexec** and other functions to execute various SQL queries, including SELECT, INSERT, UPDATE, and DELETE. To execute multiple SQL statements within a transaction, use the transaction control functionality. For example, execute SQL statements such as BEGIN, COMMIT, and ROLLBACK to control the start, commit, and rollback of transactions. Handle the errors after executing the SQL query statements.

Below is a brief example (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example):

```
/*
* After a successful connection,
* transaction blocks must be used in test instances that involve cursors.
* Putting all data in one "select * from pg_database"
* PQexec() method is too simplistic and not recommended.
 */

/* Start a transaction block. */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
 * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
 */
PQclear(res);

/*
 * Fetch data from the pg_database system catalog.
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
```

Precautions:

- The PGresult objects returned by all executed functions must be promptly released using PQclear to prevent memory leaks.
- In production environments, it is advisable to use parameterized queries or prepared statements instead of directly concatenating SQL strings.
- Avoid repeated release—each PGresult object can be released only once.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.2.4 Processing the Result Set

libpq provides functions such as **PQnfields**, **PQntuples**, and **PQfname** to help you properly parse and process the results of queries (SELECT). The general procedure is as follows:

1. Determine the execution result status.
2. Obtain the number of rows and columns in the result set.
3. Traverse and extract data.
4. Release the PGresult object (required).

Below is a brief example (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example):

```
/* First, print out the attribute name. */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* Print lines. */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

/* Release the memory of the result object to avoid memory leakage. */
PQclear(res);
```

5.5.2.5 Closing the Connection

libpq usually uses the function **PQfinish** to close the connection to the database. If the application has established multiple connections, make sure that each connection is closed correctly.

Below is a brief example (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example):

```
/* Close the portal. We do not need to check for errors. */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* End the transaction. */
res = PQexec(conn, "END");
PQclear(res);

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

Precautions:

- PQfinish must be called irrespective of connection success, failure, or exception. Failure to do so may result in connection handle leaks, which, over time, can exhaust system file descriptors or database connections.
- Be sure to release all PGresult objects (by calling PQclear) before closing connections (by calling PQfinish).
- In a multi-threaded environment, each thread should utilize a dedicated PGconn object, and thread safety must be ensured during connection closure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.3 Typical Application Development Examples

5.5.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results

NOTE

To compile the libpq source program by running **gcc**, use the **-I** *directory* option to provide the installation location of header files. (Sometimes the compiler looks for the default directory, so this option can be ignored.) Example:

```
gcc -I (Directory where header files are located) -L (Directory where the libpq library is located) -o testlibpq testlibpq.c -lpq
```

Run the following command:

```
./testlibpq
```

If a makefile is used, add the following options to variables *CPPFLAGS*, *LDFLAGS*, and *LIBS*:

```
CPPFLAGS += -I (Directory of header files)
LDFLAGS += -L (Directory of the libpq library)


LIBS += -lpq
```

Example:

```
CPPFLAGS += -I$(GAUSSHOME)/include/libpq
LDFLAGS += -L$(GAUSSHOME)/lib
```

Here is an example for the non-multi-primary mode:

```
/*
 * testlibpq.c
 * Note: testlibpq.c source program provides basic and common application scenarios of libpq.
 * The PQconnectdb, PQexec, PQntuples, and PQfinish APIs provided by libpq are used to establish database connections, execute SQL statements, obtain returned results, and clear resources.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * If users provide a value for the conninfo string in the command, that value will be used.
     * Otherwise, environment variables or all other connection parameters
     * use the default values.
```

```
    */
    if (argc > 1)
        strcpy(conninfo, argv[1]);
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Verify that the backend connection has been successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * After the connection is successful
     * Since a cursor is used in the test case, a transaction block is required.
     * Put all data in one "select * from pg_database".
     * PQexec() is too simple and is not recommended.
     */

    /* Start a transaction block. */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
     */
    PQclear(res);

    /*
     * Fetch data from the pg_database system catalog.
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);

    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* Print out the attribute name. */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* Print lines. */
    for (i = 0; i < PQntuples(res); i++)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    /* Release the memory of the result object to avoid memory leakage. */
    PQclear(res);

    /* Close the portal. Do not check for errors. */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* End the transaction. */
    res = PQexec(conn, "END");
    PQclear(res);

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

    return 0;
}
```

In the result of this example, **user_name** indicates the username of a database administrator, which varies depending on the actual environment.

```
datname       datdba       encoding      datcollate    datctype      datistemplate  datallowconn
datconnlimit  datlastsysoid  datfrozenxid  dattablespace  datcompatibilitydatacl        datfrozenxid64
datminmxid    dattimezone    dattype

template_pdb  10           7             en_US.UTF-8    en_US.UTF-8    t             t             -1
12837         0            1663          A                         3             2             PRC           P
templatea     10           7             en_US.UTF-8    en_US.UTF-8    t             f             -1
12837         0            1663          A             {=c/user_name,user_name=CTc/user_name}41372
2             PRC          D
template1     10           7             en_US.UTF-8    en_US.UTF-8    t             t             -1
12837         0            1663          A             {=c/user_name,user_name=CTc/user_name}40414
2             PRC          D
templatem     10           7             en_US.UTF-8    en_US.UTF-8    t             t             -1
12837         0            1663          M             {=c/user_name,user_name=CTc/user_name}55146
2             PRC          D
template0     10           7             en_US.UTF-8    en_US.UTF-8    t             f             -1
12837         0            1663          A             {=c/user_name,user_name=CTc/user_name}39935
2             PRC          D
postgres      10           7             en_US.UTF-8    en_US.UTF-8    f             t             -1
12837         0            1663          A                         40893         2             PRC           D
```

Below is the result of this example. The actual result may vary depending on your environment.

```
datname       datdba       encoding      datcollate    datctype      datistemplate  datallowconn
datconnlimit  datlastsysoid  datfrozenxid  dattablespace  datcompatibilitydatacl        datfrozenxid64
datminmxid    dattimezone    dattype
postgres      10           7             en_US.UTF-8    en_US.UTF-8    f             t             -1
12837         0            1663          A                         40893         2             PRC           D
```

5.5.3.2 Executing Prepared Statements

The following example creates a prepared statement with specified parameters for PQexecPrepared to execute the prepared statement.

Here is an example for the non-multi-primary mode:

```
/*
 * testlibpq2.c Test PQprepare
 */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* The values of variables such as user and passwd must be read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned. */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* Use PQconnectdb to connect to the database. connstr indicates the detailed connection information. */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, passwd);

    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }

    /* Create table t01. */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01 values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }

    /* cmd_sql query */
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    /* Parameter corresponding to $1 in cmd_sql */
    paramTypes[0] = 23;
    /* PQprepare creates a prepared statement with given parameters. */
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
  if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
        /* Execute the prepared statement. */
        res = PQexecPrepared(conn,
                             "pre_name",
                             1,
                             paramValues,
                             paramLengths,
                             paramFormats,
                             0);
        if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
        {
            printf("%s\n",PQerrorMessage(conn));
            PQclear(res);
            PQfinish(conn);
            return -1;
        }
        cnt = PQntuples(res);
        printf("return %d rows\n", cnt);
        for (i=0; i<cnt; i++)
        {
            printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
        }
        PQclear(res);
    }
    /* The execution is complete. Close the connection. */
    PQfinish(conn);
    return 0;
}
```

Below are the results of the preceding two examples:

```
Connect database success!
NOTICE:  table "t01" does not exist, skipping
return 1 rows
row 0: 23
return 1 rows
row 0: 23
```

5.5.3.3 Binding Parameters and Returning a Binary Result

The following example runs a command to bind parameters and requests the query result in binary format:

```
CREATE TABLE test1 (i int4, t text);
INSERT INTO test1 values (2, 'ho there');
```

After the database is filled by the commands above, run the following example:

Here is an example for the non-multi-primary mode:

```
/*
 * testlibpq3.c
 * Test PQexecParams.
 *
 * The expected output is as follows:
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * This function is used to print out the query results. The results are in binary format
* and fetched from the table created in the comment above.
 */
static void show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* Obtain the field value. (Ignore the possibility that they may be null.) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * The binary representation of TEXT is text, so libpq can append a zero byte to it,
         *  and think of it as a C string.
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int main(int argc, char **argv)
{
    /* The values of variables such as user and passwd are read from environment variables or configuration
files. When using environment variables, users must configure them manually. In other cases, users can
assign values directly as character strings. */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
```

```
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * If users provide a parameter in the command,
     * its value will be used as the conninfo character string. Otherwise,
     * environment variables or default values will be used instead.
     */
    if (argc > 1)
        strcpy(conninfo, argv[1]);
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* Establish a database connection. */
    conn = PQconnectdb(conninfo);

     /* Check whether the connection to the server was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    /* Convert the integer value "2" to the network byte order. */
    binaryIntVal = htonl((uint32_t) 2);

    /* Set the parameter array for PQexecParams. */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* Binary */

    /* PQexecParams executes a command with parameters. */
    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* One parameter */
                       NULL,    /* Enable the backend to deduce the parameter type. */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* Binary result is required. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    /* Display the binary result. */
    show_binary_results(res);

    PQclear(res);

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

    return 0;
}
```

Below are the results of the preceding two examples:

```
tuple 0: got
 i = (4 bytes) 2
 t = (8 bytes) 'ho there'
```

5.5.4 libpq API Reference

NOTICE

Calls to the PQfn API using libpq is not supported yet.

5.5.4.1 Overview

The libpq API reference provides a set of core functions required for developing C-language database clients. These functions are categorized by functionality and cover the entire development workflow—from connection establishment, SQL execution, and asynchronous processing to result set processing.

You can consult relevant categories based on your specific needs to quickly locate the required functions. Note that calling the PQfn API via libpq is currently not supported. This section details only the most commonly used APIs. To explore other APIs, click **here**. The libpq API reference is structured into the following six parts:

- **Database Connection Control Functions**:

  Database connection control functions deal with connections to the database server. An application can have several server connections open at one time. (One reason to do that is to access more than one database.) Each connection is represented by a PGconn object, which is obtained from the PQconnectdb, PQconnectdbParams, or PQsetdbLogin function. Alternatively, a connection object may be acquired using the PQconnectStart API in combination with the asynchronous PQconnectPoll method. Note that these functions will always return a non-null object pointer unless memory allocation fails, in which case a null pointer is returned. The connection establishment API is stored in the PGconn object. The PQstatus function can be called to check the return value for a successful connection. Since PGconn stores SSL context in thread-local storage, the PGconn object must be released by the same thread that allocated it.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Database Statement Execution Functions**:

  Once a connection to the database server is successfully established, the functions described in this section are used to execute SQL queries and commands. They represent the most frequently used APIs in libpq.

- **Asynchronous Command Processing Functions**:

  Asynchronous command processing functions enable non-blocking SQL execution, making them suitable for high-concurrency, low-latency, or multi-command scenarios. These functions allow applications to send SQL commands, poll command execution status, and perform other operations without blocking the current thread.

- **Query Cancellation Functions**:

  Query cancellation functions are used to proactively terminate a running SQL statement during query execution. They are primarily applied in scenarios such as timeout control for long-running queries, user-initiated cancellation, and rapid termination under exceptional conditions.

- **Functions for Processing Database Result**:

  Database result processing functions are used to parse and extract the result sets (PGresult) returned by PQexec or asynchronous execution. These functions provide capabilities such as retrieving row/column counts, extracting field values, and checking for null values.

Together, these functions constitute the complete functional framework of libpq.

5.5.4.2 Database Connection Control Functions

5.5.4.2.1 PQconnectdbParams

Description

Establishes a new connection with the database server.

Prototype

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

Parameters

Table 5-78 PQconnectdbParams parameters

Keyword	Description
keywords	An array of strings, each of which is a keyword.
values	Value assigned to each keyword.
expand_dbname	When **expand_dbname** is non-zero, the **dbname** keyword value can be recognized as a connection string. Only **dbname** that first appears is treated in this way, and any subsequent **dbname** value is treated as a database name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return Values

PGconn * points to the object that contains the connection. The memory is allocated inside the function.

Precautions

This function establishes a new database connection using the parameters taken from two NULL-terminated arrays. Unlike PQsetdbLogin, the parameter set can be extended without changing the function signature. Therefore, use of this function (or its non-blocking analogs PQconnectStartParams and PQconnectPoll) is preferred for new application programming.

5.5.4.2.2 PQconnectdb

Description

Establishes a new connection with the database server.

Prototype

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

Parameters

Table 5-79 PQconnectdb parameters

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

Return Values

PGconn * points to the object that contains the connection. The memory is allocated inside the function.

Precautions

- This function opens a new connection with the parameters specified by **conninfo**.
- The input parameter can be empty, indicating that all default parameters can be used. It can contain one or more values separated by spaces or contain a URL.

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.2.3 PQconninfoParse

Description

Returns parsed connection options based on the connection.

Prototype

```
PQconninfoOption* PQconninfoParse(const char* conninfo, char** errmsg);
```

Parameters

Table 5-80

Keyword	Description
conninfo	Passed string. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URL.
errmsg	Error information.

Return Values

PQconninfoOption pointers.

5.5.4.2.4 PQconnectStart

Description

Establishes a non-blocking connection with the database server.

Prototype

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

Parameters

Table 5-81

Keyword	Description
conninfo	String of connection information. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URL.

Return Values

PGconn pointers.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.2.5 PQerrorMessage

Description

Returns error information on a connection.

Prototype

```
char* PQerrorMessage(const PGconn* conn);
```

Parameters

Table 5-82

Keyword	Description
conn	Connection handle.

Return Values

Char pointers.

5.5.4.2.6 PQsetdbLogin

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQsetdbLogin(const char* pghost, const char* pgport, const char* pgoptions, const char* pgtty,
    const char* dbName, const char* login, const char* pwd);
```

Parameters

Table 5-83 PQsetdbLogin parameters

Keyword	Description
pghost	Name of the host to be connected. For details, see the **host** column described in **Connection Parameters**.
pgport	Port number of the host server. For details, see the **port** column described in **Connection Parameters**.
pgoptions	Command-line options to be sent to the server during running. For details, see the **options** column described in **Connection Parameters**.
pgtty	Output direction of server logs. Currently, this setting does not take effect.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
dbName	Name of the database to connect to. For details, see the **dbname** column described in **Connection Parameters**.
login	Username for connection. For details, see the **user** column described in **Connection Parameters**.
pwd	Password used for authentication during connection. For details, see the **password** column described in **Connection Parameters**.

Return Values

PGconn * points to the object that contains the connection. The memory is allocated inside the function.

Precautions

- This function is the predecessor of PQconnectdb with a fixed set of parameters. When an undefined parameter is called, its default value is used. The default value can be **NULL** or an empty string.
- If the **dbName** value contains an equal sign (=) or a valid prefix in the connection URL, it is taken as a conninfo string and passed to PQconnectdb, and the other parameters are consistent with those of PQconnectdbParams.

5.5.4.2.7 PQfinish

Description

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

Prototype

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

Parameters

Table 5-84 PQfinish parameters

Keyword	Description
conn	Object pointer that contains the connection information.

Precautions

If the server connection attempt fails (as indicated by PQstatus), the application should call PQfinish to release the memory used by the PGconn object. The PGconn pointer must not be used again after PQfinish has been called.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Typical Application Development Examples**.

5.5.4.2.8 PQreset

Description

Resets the communication port to the server.

Prototype

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

Parameters

Table 5-85 PQreset parameters

Keyword	Description
conn	Object pointer that contains the connection information.

Precautions

This function will close the connection to the server and attempt to establish a new connection to the same server by using all the parameters previously used. This function is applicable to fault recovery after a connection exception occurs.

5.5.4.2.9 PQstatus

Description

Returns the connection status.

Prototype

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

Parameters

Table 5-86 PQ status parameters

Keyword	Description
conn	Object pointer that contains the connection information.

Return Values

ConnStatusType indicates the connection status. The enumerated values are as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CONNECTION_STARTED
Waiting for the connection to be established.

CONNECTION_MADE
Connection succeeded; waiting to send.

CONNECTION_AWAITING_RESPONSE
Waiting for a response from the server.

CONNECTION_AUTH_OK
Authentication received; waiting for backend startup to complete.

CONNECTION_SSL_STARTUP
Negotiating SSL encryption.

CONNECTION_SETENV
Negotiating environment-driven parameter settings.

CONNECTION_OK
Normal connection.

CONNECTION_BAD
Failed connection.
```

Precautions

The connection status can be one of the preceding values. After the asynchronous connection procedure is complete, only two of them, **CONNECTION_OK** and **CONNECTION_BAD**, can return. **CONNECTION_OK** indicates that the connection to the database is normal. **CONNECTION_BAD** indicates that the connection to the database fails. Generally, the **CONNECTION_OK** state remains until PQfinish is called. However, a communication failure may cause the connection status to turn to **CONNECTION_BAD** before the connection procedure is complete. In this case, the application can attempt to call PQreset to restore the communication.

Examples

Refer to **Typical Application Development Examples**.

5.5.4.3 Database Statement Execution Functions

5.5.4.3.1 PQexec

Description

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

Prototype

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-87 PQexec parameters

Keyword	Description
conn	Object pointer that contains the connection information.
query	Query string to be executed.

Return Values

PGresult indicates the object pointer that contains the query result.

Precautions

The PQresultStatus function should be called to check the return value for any errors (including the value of a null pointer, in which **PGRES_FATAL_ERROR** will be returned). The PQerrorMessage function can be called to obtain more information about such errors.

NOTICE

The command string can contain multiple SQL commands separated by semicolons (;). Multiple queries sent in a PQexec call are processed in one transaction, unless there are specific BEGIN/COMMIT commands in the query string to divide the string into multiple transactions. Note that the returned PGresult structure describes only the result of the last command executed from the string. If a command fails, the string processing stops and the returned PGresult describes the error condition.

The PQexec function is adequate for submitting commands in normal, synchronous applications. It has a few deficiencies, however, that can be of importance to some users:

- PQexec waits for the command to be completed. The application might have other work to do (such as maintaining a user interface), in which case it will not want to block waiting for the response.
- Since the execution of the client application is suspended while it waits for the result, it is hard for the application to decide that it would like to try to cancel the ongoing command.
- PQexec can return only one PGresult structure. If the submitted command string contains multiple SQL commands, all but the last PGresult are discarded by PQexec.
- PQexec always collects the command's entire result, buffering it in a single PGresult. While this simplifies error-handling logic for the application, it can be impractical for results containing many rows.

Applications that do not like these limitations can instead use the underlying functions that PQexec is built from:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PQsendQuery and PQgetResult. PQsendQueryParams, PQsendPrepare, and PQsendQueryPrepared can be used together with PQgetResult.

Examples

Refer to **Typical Application Development Examples**.

5.5.4.3.2 PQprepare

Description

Commits a request to create a prepared statement with given parameters and waits for completion.

Prototype

```
PGresult* PQprepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

Parameters

Table 5-88 PQprepare parameters

Keyword	Description
conn	Object pointer that contains the connection information.
stmtName	Name of **prepare** to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Values

PGresult indicates the object pointer that contains the query result.

Precautions

- This function creates a prepared statement for later execution with PQexecPrepared. It allows commands to be repeatedly executed, without being parsed and planned each time they are executed.
- PQprepare is supported only in protocol 3.0 or later. It will fail when protocol 2.0 is used.
- This function creates a prepared statement named **stmtName** from the query string, which must contain an SQL command. **stmtName** can be **""** to create an unnamed statement. In this case, any pre-existing unnamed statement will be automatically replaced. Otherwise, this is an error if the statement name has been defined in the current session. If any parameters are used, they are referred to in the query as $1, $2, and so on. **nParams** is the number of parameters for which types are pre-specified in the array paramTypes[]. (The

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

array pointer can be **NULL** when **nParams** is **0**.) paramTypes[] specifies the data types to be assigned to the parameter symbols by OID. If **paramTypes** is **NULL**, or any element in the array is **0**, the server assigns a data type to the parameter symbol in the same way as it does for an untyped literal string. In addition, the query can use parameter symbols whose numbers are greater than **nParams**. Data types of these symbols will also be inferred.

NOTICE

You can also execute the **SQLPREPARE** statement to create a prepared statement that is used with PQexecPrepared. Although there is no libpq function of deleting a prepared statement, the SQL DEALLOCATE statement can be used for this purpose.

Examples

Refer to **Typical Application Development Examples**.

5.5.4.3.3 PQclear

Description

Releases the storage associated with PGresult. Any query result should be released by PQclear when it is no longer needed.

Prototype

```
void PQclear(PGresult* res);
```

Parameters

Table 5-89 PQclear parameters

Keyword	Description
res	Object pointer that contains the query result.

Precautions

PGresult is not automatically released. That is, it does not disappear when a new query is committed or even if you close the connection. To delete it, you must call PQclear. Otherwise, memory leakage occurs.

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.3.4 PQfnumberEx

Description

Returns the column number associated with the given column name. Columns are numbered starting at 0.

Prototype

```
int PQfnumberEx(const PGresult* res, const char* field_name, bool case_sensitive);
```

Parameters

Table 5-90 PQfnumberEx parameters

Keyword	Description
res	Handle to the operation result.
field_name	Column name.
case_sensitive	Specifies whether the column name is case-sensitive.

Return Values

Integers of the int type.

5.5.4.3.5 PQexecParams

Description

Runs a command to bind parameters.

Prototype

```
PGresult* PQexecParams(PGconn* conn,
                       const char* command,
                       int nParams,
                       const Oid* paramTypes,
                       const char* const* paramValues,
                       const int* paramLengths,
                       const int* paramFormats,
                       int resultFormat);
```

Parameters

Table 5-91 PQexecParams parameters

Keyword	Description
conn	Connection handle.
command	SQL text string.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
nParams	Number of parameters to be bound.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter length. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Values

PGresult pointers.

5.5.4.3.6 PQexecParamsBatch

Description

Runs a command to bind batches of parameters.

Prototype

```
PGresult* PQexecParamsBatch(PGconn* conn,
                            const char* command,
                            int nParams,
                            int nBatch,
                            const Oid* paramTypes,
                            const char* const* paramValues,
                            const int* paramLengths,
                            const int* paramFormats,
                            int resultFormat);
```

Parameters

Table 5-92 PQexecParamsBatch parameters

Keyword	Description
conn	Connection handle.
command	SQL text string.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
nParams	Number of parameters to be bound.
nBatch	Number of batch operations.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter length. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Values

PGresult pointers.

5.5.4.3.7 PQexecPrepared

Description

Sends a request to execute a prepared statement with given parameters and waits for the result.

Prototype

```
PGresult* PQexecPrepared(PGconn* conn,
                         const char* stmtName,
                         int nParams,
                         const char* const* paramValues,
                         const int* paramLengths,
                         const int* paramFormats,
                         int resultFormat);
```

Parameters

Table 5-93 PQexecPrepared parameters

Keyword	Description
conn	Connection handle.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
paramValues	Actual values of parameters.
paramLengths	Actual data lengths of parameters. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

NOTE

When binding character data that includes the '\0' character and setting the libpq parameter **supportZeroCharacter** to **1**, ensure that the length of the input string (**paramLengths**) is equal to the actual length of the string with the '\0' character. If the input string is longer than the string with the '\0' character, out-of-bounds memory access may occur, resulting in a coredump on the libpq side. If the input string is shorter than the string with the '\0' character, the latter will be truncated based on the length of the former.

When **supportZeroCharacter** is set to **1**, also ensure that the character set of the client is the same as that of the database.

Return Values

PGresult pointers.

5.5.4.3.8 PQexecPreparedBatch

Description

Sends a request to execute a prepared statement with batches of given parameters and waits for the result.

Prototype

```
PGresult* PQexecPreparedBatch(PGconn* conn,
                              const char* stmtName,
                              int nParams,
                              int nBatchCount,
                              const char* const* paramValues,
                              const int* paramLengths,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                const int* paramFormats,
                int resultFormat);
```

Parameters

Table 5-94 PQexecPreparedBatch parameters

Keyword	Description
conn	Connection handle.
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
nBatchCount	Number of batches.
paramValues	Actual values of parameters.
paramLengths	Actual data lengths of parameters. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

Return Values

PGresult pointers.

5.5.4.4 Asynchronous Command Processing Functions

5.5.4.4.1 PQsendQuery

Description

Commits a command to the server without waiting for the result. If the query is successful, **1** is returned. Otherwise, **0** is returned.

Prototype

```
int PQsendQuery(PGconn* conn, const char* query);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-95 PQsendQuery parameters

Keyword	Description
conn	Object pointer that contains the connection information.
query	Query string to be executed.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

After PQsendQuery is successfully called, call PQgetResult one or more times to obtain the results. PQsendQuery cannot be called again (on the same connection) until PQgetResult returns a null pointer, indicating that the command execution is complete.

5.5.4.4.2 PQsendQueryParams

Description

Commits a command with separated parameters to the server without waiting for the result.

Prototype

```
int PQsendQueryParams(PGconn* conn, const char* command, int nParams, const Oid* paramTypes,
    const char* const* paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```

Parameters

Table 5-96 PQsendQueryParams parameters

Keyword	Description
conn	Object pointer that contains the connection information.
command	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Parameter type.
paramValues	Parameter value.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
paramLengths	Parameter length. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

This function is equivalent to PQsendQuery. The only difference is that query parameters can be specified separately from the query string. The parameter processing of this function is similar to that of PQexecParams. It cannot work on connections using protocol v2.0, and it allows only one command to appear in the query string.

5.5.4.4.3 PQsendPrepare

Description

Sends a request to create a prepared statement with given parameters, without waiting for completion.

Prototype

```
int PQsendPrepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

Parameters

Table 5-97 PQsendPrepare parameters

Keyword	Description
conn	Object pointer that contains the connection information.
stmtName	Name of **prepare** to be executed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

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

5.5.4.4.4 PQsendQueryPrepared

Description

Sends a request to execute a prepared statement with given parameters, without waiting for the result.

Prototype

```
int PQsendQueryPrepared(PGconn* conn, const char* stmtName, int nParams, const char* const*
paramValues,
    const int* paramLengths, const int* paramFormats, int resultFormat);
```

Parameters

Table 5-98 PQsendQueryPrepared parameters

Keyword	Description
conn	Object pointer that contains the connection information.
stmtName	Name of a prepared statement to be executed.
nParams	Parameter quantity.
paramValues	Parameter value.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Description
paramLengths	Parameter length. **NOTICE** In the following cases, the value of **paramLengths** is forcibly applied to parameters: • Parameter values are in binary format. • Parameter values are in text format, and the connection parameter **force_no_truncation** or **supportZeroCharacter** is set to **1**. In these cases, **paramLengths** cannot be empty, and the caller must ensure the correctness of parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

This function is similar to PQsendQueryParams, but the command to be executed is specified by naming a previously-prepared statement, instead of providing a query string. The parameters of this function are processed in the same way as PQexecPrepared. Like PQexecPrepared, it cannot work on connections using protocol v2.0.

5.5.4.4.5 PQflush

Description

Tries to flush any queued output data to the server.

Prototype

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

Parameters

Table 5-99 PQflush parameters

Keyword	Description
conn	Object pointer that contains the connection information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return Values

int indicates the operation result. If the operation is successful (or the send queue is empty), **0** is returned. If the operation fails, **–1** is returned. If all data in the send queue fails to be sent, **1** is returned. (This case occurs only when the connection is non-blocking.) The failure cause is stored in **conn->errorMessage**.

Precautions

Call PQflush after sending any command or data over a non-blocking connection. If **1** is returned, wait for the socket to become read- or write-ready. If the socket becomes write-ready, call PQflush again. If the socket becomes read-ready, call PQconsumeInput and then call PQflush again. Repeat the operation until the value **0** is returned for PQflush. It is necessary to check read-ready and use PQconsumeInput to exhaust input because the server may prevent attempts to send data (such as NOTICE messages) to the client and does not read the client's data until the client reads its data. Once PQflush returns **0**, wait for the socket to be read-ready and then read the response as described above.

5.5.4.5 Query Cancellation Functions

5.5.4.5.1 PQgetCancel

Description

Creates a data structure that contains the information required to cancel a command issued through a specific database connection.

Prototype

```
PGcancel* PQgetCancel(PGconn* conn);
```

Parameters

Table 5-100 PQgetCancel parameters

Keyword	Description
conn	Object pointer that contains the connection information.

Return Values

PGcancel points to the object that contains the cancel information.

Precautions

PQgetCancel creates a PGcancel object for a given PGconn connection object. If the given connection object (**conn**) is NULL or an invalid connection, it will return NULL. The PGcancel object is an opaque structure that cannot be directly accessed by applications. It can be transferred only to PQcancel or PQfreeCancel.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.5.2 PQfreeCancel

Description

Releases the data structure created by PQgetCancel.

Prototype

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

Parameters

Table 5-101 PQfreeCancel parameters

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

Precautions

PQfreeCancel releases a data object previously created by PQgetCancel.

5.5.4.5.3 PQcancel

Description

Requests the server to abandon processing of the current command.

Prototype

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

Parameters

Table 5-102 PQcancel parameters

Keyword	Description
cancel	Points to the object pointer that contains the cancel information.
errbuf	Buffer for storing error information.
errbufsize	Size of the buffer for storing error information.

Return Values

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **errbuf**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- Successful sending does not guarantee that the request will have any effect. If the cancellation is valid, the current command is terminated early and an error is returned. If the cancellation fails (for example, because the server has processed the command), no result is returned.
- If **errbuf** is a local variable in a signal handler, you can safely call PQcancel from the signal handler. For PQcancel, the PGcancel object is read-only, so it can also be called from a thread that is separate from the thread that is operating on the PGconn object.

5.5.4.6 Functions for Processing Database Result

5.5.4.6.1 PQgetvalue

Description

Returns a single threshold for a row of PGresult. Row and column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQgetvalue(const PGresult* res, int tup_num, int field_num);
```

Parameters

Table 5-103 PQgetvalue parameters

Keyword	Description
res	Handle to the operation result.
tup_num	Number of rows.
field_num	Number of columns.

Return Values

For data in text format, PQgetvalue returns a null-terminated string representation of the threshold.

For binary data, the value is a binary representation determined by the typsend and typreceive functions of the data type.

If the threshold is empty, an empty string is returned.

5.5.4.6.2 PQnfields

Description

Returns the number of columns (fields) in each row of the query result.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Prototype

```
int PQnfields(const PGresult* res);
```

Parameters

Table 5-104 PQnfields parameters

Keyword	Description
res	Handle to the operation result.

Return Values

Int numbers.

5.5.4.6.3 PQntuples

Description

Returns the number of rows (tuples) in the query result.

Prototype

```
int PQntuples(const PGresult* res);
```

Parameters

Table 5-105 PQntuples parameters

Keyword	Description
res	Handle to the operation result.

NOTICE

PQntuples returns an integer result. An overflow may occur if the return value is out of the value range allowed in a 32-bit OS.

Return Values

Int numbers.

Examples

Refer to **Typical Application Development Examples**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.5.4.6.4 PQfname

Description

Returns the column name associated with the given column number. Column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQfname(const PGresult* res, int field_num);
```

Parameters

Table 5-106 PQfname parameters

Keyword	Description
res	Handle to the operation result.
field_num	Number of columns.

Return Values

Char pointers.

5.5.4.6.5 PQresultStatus

Description

Returns the result status of a command.

Prototype

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

Parameters

Table 5-107 PQresultStatus parameters

Keyword	Description
res	Object pointer that contains the query result.

Return Values

PQresultStatus indicates the command execution status. The enumerated values are as follows:

```
PQresultStatus can return one of the following values:
PGRES_EMPTY_QUERY
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The string sent to the server was empty.

PGRES_COMMAND_OK
A command that does not return data was successfully executed.

PGRES_TUPLES_OK
A query (such as SELECT or SHOW) that returns data was successfully executed.

PGRES_COPY_OUT
The data copied from the server begins to transmit.

PGRES_COPY_IN
The data copied to the server begins to transmit.

PGRES_BAD_RESPONSE
The response from the server cannot be understood.

PGRES_NONFATAL_ERROR
A non-fatal error (notification or warning) occurred.

PGRES_FATAL_ERROR
A fatal error occurred.

PGRES_COPY_BOTH
The data copied to and from the server begins to transmit. This state occurs only in streaming replication.

PGRES_SINGLE_TUPLE
PGresult contains a result tuple from the current command. This state occurs in a single-row query.

Precautions

- Note that the SELECT command that happens to retrieve zero rows still returns **PGRES_TUPLES_OK**. **PGRES_COMMAND_OK** is used for commands that can never return rows (such as INSERT or UPDATE, without return clauses). The result status **PGRES_EMPTY_QUERY** might indicate a bug in the client software.
- The result status **PGRES_NONFATAL_ERROR** will never be returned directly by PQexec or other query execution functions. Instead, such results will be passed to the notice processor.

Examples

Refer to **Typical Application Development Examples**.

5.6 Development Based on Psycopg

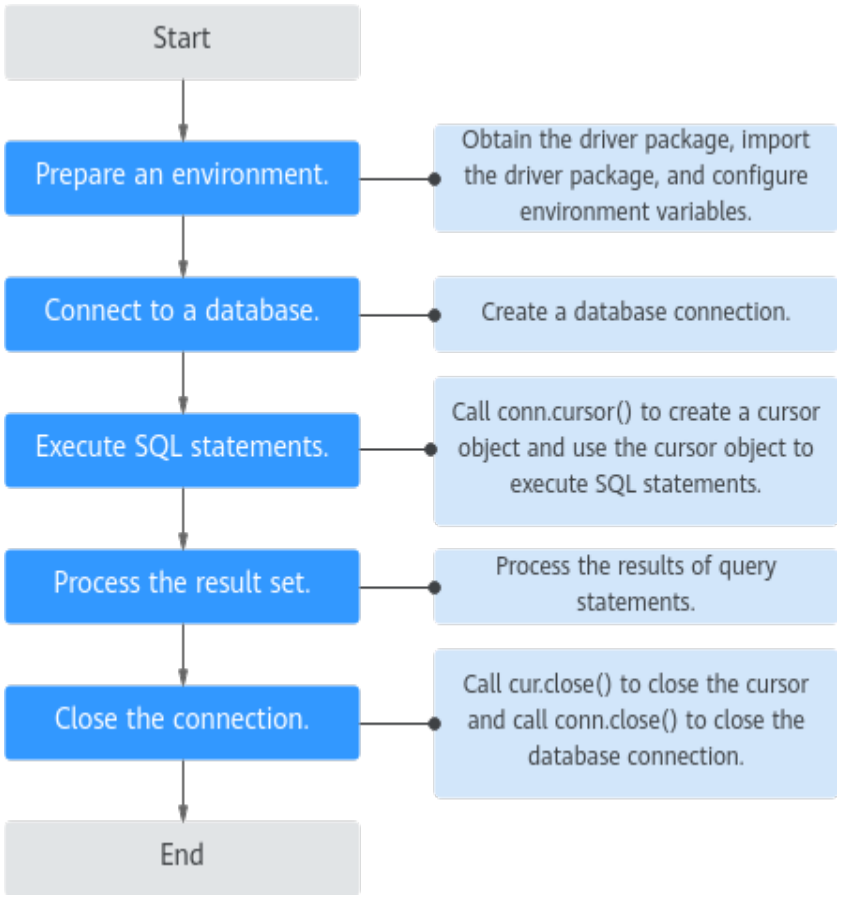
5.6.1 Development Process

The application development process based on psycopg2 involves these steps: prepare an environment, connect to a database, execute SQL statements, process the result set, and close the connection.

Figure 5-18 shows the development process.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-18 Application development process based on psycopg2



5.6.2 Development Procedure

5.6.2.1 Preparing an Environment

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 5-108**.

Table 5-108 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

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

- Verifying the software package integrity on a Linux server:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

a. Upload the software package and verification package to the same directory on the VM.

b. Run the following command to verify the integrity of the software package:

```
F=$(echo DBS-GaussDB-driver*.tar.gz);echo "$(awk '{print $1}' "$F.sha256")  $F" | sha256sum -c
-
```

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

```
DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz: OK
```

- Verifying the software package integrity on a Windows server:

a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.

b. Run the following command to obtain the hash value of the driver package:

certutil -hashfile *{local_directory_of_the_driver_package}**{driver_package_name}* **sha256**

- Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
- Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz**.

Example: **certutil -hashfile** *C:|Users**DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz* **sha256**

c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-108**.

- If they are consistent, the verification is successful.
- If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining the Python Driver Package

The following describes how to obtain the Python driver package:

Step 1 Decompress the obtained driver package, which includes various driver packages. Follow the steps below to find the file path for the specific package you need. The following is the standard path hierarchy for a driver package:

DBS-GaussDB-driver_*/*Database deployment mode***/***Driver type***/***OS architecture***/***Release package***/***Driver package*

Step 2 Select a database deployment mode.

Select **Distributed** for the distributed deployment mode.

Step 3 Select a driver type.

Select **python_driver.tar.gz** and decompress it.

Step 4 Select an OS architecture.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

You need to select the driver package of the corresponding OS architecture based on the OS of the host where the client is located.

Step 5 Select the Python release package.

Step 6 Decompress the Python release package to obtain the driver packages in .tar.gz and .whl formats. The driver packages in the two formats have the same functions. Select a driver package based on your service requirements.

- Driver package formats in non-multi-primary mode:

  GaussDB-Kernel_*Database version number*_**Python**_*CPU architecture*_*Python version*_**Distributed.tar.gz**

  > **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Python.tar.gz**

  > **GaussDB_Kernel**_*Database version number-Psycopg2 version number-Python version number*-**abiTag**-*Supported platform*.**whl**

  Example:

  ```
  GaussDB-Kernel_507.0.0.B071_Python_X86_Py3.11_Distributed.tar.gz
  > GaussDB-Kernel_507.0.0_Euler_64bit_Python.tar.gz
  > GaussDB_Kernel_507_0_0-2.9.10-py311-none-linux_x86_64.whl
  ```

  NOTE

  - The .tar.gz driver package contains the following folders after decompression:
    - **psycopg2**: source code file of the Python driver.
    - **lib**: dependency library of the Python driver.
  - After the .whl driver package is installed, the following folders are generated in the **site-packages** directory of the corresponding Python environment:
    - **psycopg2**: source code file of the Python driver.
    - **gaussdb_psycopg2.lib**: dependency library of the Python driver.
  - The .whl driver package follows these naming conventions:
    - *abiTag*: corresponding to **{abi tag}**, which represents the tag of the application binary interface (ABI), such as **cp311** or **none**.
    - *Supported platform*: corresponding to **{platform tag}**, which represents the supported platform. For example, **linux_x86_64** represents the Linux x86_64 platform, while **any** indicates platform independence.
  - Constraints for using the .whl driver package: When installing the .whl driver package for Python, ensure that the Python version in the environment is the same as *Python version number* in the .whl driver package name.
  - Currently, the Python driver package is limited to Python 3.11.

----End

Performing Pre-installation Checks

Before installing the psycopg2 Python driver, check whether an open-source version of the driver is already installed in your environment. If present, remove it to ensure proper functionality of the target driver. Follow the check steps below:

Step 1 Switch to the **root** user and run the following command to check whether the psycopg2 Python driver is already installed in your environment:

```
ll $(python3 -c 'import site; print(site.getsitepackages()[0])') | grep psycopg2
```

If the corresponding directory appears in the output, proceed to **the next step**. Otherwise, the check is complete.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 2 Run the following command to remove the installed psycopg2 driver file:

```
rm -rf $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2*
```

----End

Importing the .tar.gz driver package and Configuring Environment Variables

Step 1 Import the driver package **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Python.tar.gz** to the root directory of your environment.

Step 2 Decompress the driver package.

```
tar zxvf xxxx_Python.tar.gz
```

Step 3 Switch to the **root** user and copy files from the **psycopg2** folder to the **site-packages** folder within the Python installation directory.

```
su root
cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
```

NOTICE

The purpose of the **$(python3 -c 'import site; print(site.getsitepackages()[0])')** command is to obtain the main path of the **site-packages** directory where Python installs third-party packages. If an error message is displayed indicating that the directory does not exist, run the **python3 -c "import site; print(site.getsitepackages())"** command to list all **site-packages** paths and select a valid one for replacement.

Step 4 Change the permission of **psycopg2** to **755**.

```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```

Step 5 Configure **psycopg2** in the environment variable *PYTHONPATH* and put it into effect.

```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```

Step 6 (Non-database users) Configure *[path-to-lib]* (absolute path of the post-decompression **lib** folder) in the environment variable *LD_LIBRARY_PATH*.

```
export LD_LIBRARY_PATH=[path-to-lib]:$LD_LIBRARY_PATH
```

NOTE

Database users do not need to manually configure *LD_LIBRARY_PATH*.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Importing the .whl driver package and Configuring Environment Variables

NOTICE

Before installing the .whl driver package, check whether the database has installed the driver package **GaussDB_Kernel**_*Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform*.**whl**. If this package is present and has not been uninstalled using the **pip3 uninstall** or **python3 -m pip uninstall** command, you are advised to uninstall it first by running the **pip3 uninstall** or **python3 -m pip uninstall** command. For detailed instructions, refer to **Method 2**.

Step 1 Import the driver package **GaussDB_Kernel**_*Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform*.**whl** to the root directory of your environment.

Step 2 Switch to the **root** user and install the .whl driver package.

```
su root
```

If your environment supports **pip**, run the following command:

```
pip3 install GaussDB_Kernel_Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform.whl
```

If your environment does not support **pip**, run the following command:

```
python3 -m pip install GaussDB_Kernel_Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform.whl
```

Step 3 (Optional) Run the following command to grant non-root users the permission to use the Python driver:

```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```

NOTICE

The purpose of the **$(python3 -c 'import site; print(site.getsitepackages()[0])')** command is to obtain the main path of the **site-packages** directory where Python installs third-party packages. If an error message is displayed indicating that the directory does not exist, run the **python3 -c "import site; print(site.getsitepackages())"** command to list all **site-packages** paths and select a valid one for replacement.

Step 4 Configure environment variables.

- Non-database installation users: After the .whl driver package is installed, configure the **lib** directory in the environment variable *LD_LIBRARY_PATH*.

  ```
  lib_path=$(python3 -c 'import site; print(site.getsitepackages()[0]+"/gaussdb_psycopg2.lib")')
  export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH
  ```

- Database installation users: There is no need to manually configure the environment variable *LD_LIBRARY_PATH*.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

(Optional) Uninstalling the Driver Package

Step 1 Switch to the **root** user.

```
su root
```

Step 2 Uninstall the Python driver.

- Method 1: This method is recommended if the Python driver was installed from the .tar.gz driver package. While it can also be used if the Python driver was installed using the .whl driver package, **Method 2** is preferred in that case.

  ```
  site_package_path=$(python3 -c 'import site; print(site.getsitepackages()[0])')
  rm -rf $site_package_path/psycopg2
  ```

  NOTICE

  The purpose of the **site_package_path=$(python3 -c 'import site; print(site.getsitepackages()[0])')** command is to obtain the main path of the **site-packages** directory where Python installs third-party packages. If an error message is displayed indicating that the directory does not exist, run the **python3 -c "import site; print(site.getsitepackages())"** command to list all **site-packages** paths and select a valid one for replacement.

  - To uninstall the .whl driver package, run the following command:

    ```
    rm -rf $site_package_path/gaussdb_psycopg2.lib $site_package_path/GaussDB_Kernel_*.dist-info
    ```

  - To uninstall the .tar.gz driver package, run the following command:

    ```
    rm -rf [path-to-lib]
    ```

    [path-to-lib] indicates the location where you chose to store the **lib** directory when installing the .tar.gz driver package.

- Method 2: This method is specifically applicable to scenarios where the Python driver was installed using the .whl driver package.

  - If the .whl driver package was installed using **pip install**, run the following command:

    ```
    pip3 uninstall GaussDB_Kernel_Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform.whl
    ```

  - If the .whl driver package was installed using **python3 -m pip install**, run the following command:

    ```
    python3 -m pip uninstall GaussDB_Kernel_Database version number-Psycopg2 version number-Python version number-abiTag-Supported platform.whl
    ```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.6.2.2 Connecting to a Database

NOTICE

- If the Python driver version is 2.9.10 or later, the client does not support certain encoding sets (such as ZHS16GBK and GB18030_2022). In this case:
  - The connection parameter **client_encoding** cannot be set to these encoding sets.
  - You cannot connect to databases using any of these encoding sets.
- To determine your Python driver version, run the following command:
  ```
  python3 -c "import psycopg2;print(psycopg2.__version__)"
  ```

5.6.2.2.1 Connection Methods

The psycopg2.connect function is available for establishing database connections. See **psycopg2.connect()** for the function parameters. When using the psycopg2.connect function to connect to a database, you can pass a character string via the **dsn** parameter or pass multiple keyword parameters via ****kwargs**.

- Method 1: psycopg2.connect(dsn). This method leverages the **dsn** parameter to pass a character string that includes the database name, username, password, server address, and port number. For detailed steps, refer to **Connection via dsn**.
- Method 2: psycopg2.connect(**kwargs). This method leverages ****kwargs** to pass multiple keyword parameters—the database name, username, password, server address, and port number. For detailed steps, refer to **Connection via **kwargs**.

5.6.2.2.2 Connection in Non-encrypted Mode

Connection via dsn

Call psycopg2.connect(dsn) to create a database connection, as follows:

Step 1 Import the psycopg2 and os modules.

```
import psycopg2
import os
```

Step 2 Specify the username and password. It is advisable to store the username and password in environment variables because writing them directly in the code can create security risks.

```
userName = os.getenv('EXAMPLE_USERNAME_ENV')
passWord = os.getenv('EXAMPLE_PASSWORD_ENV')
```

Step 3 Specify **host** and **port**. It is advisable to store **host** and **port** in environment variables.

```
host= os.getenv('EXAMPLE_HOST_ENV')
port= os.getenv('EXAMPLE_PORT_ENV')
```

Step 4 Call psycopg2.connect(dsn) to create a database connection.

- In non-multi-primary mode, you need to specify the database name, server address, and port number.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
conn = psycopg2.connect(f"dbname=database user={userName} password={passWord} host={host}
port={port}")
```

----End

Connection via **kwargs

Call psycopg2.connect(**kwargs) to create a database connection, as follows:

Step 1 Import the psycopg2 and os modules.

```
import psycopg2
import os
```

Step 2 Specify the username and password. It is advisable to store the username and password in environment variables because writing them directly in the code can create security risks.

```
userName = os.getenv('EXAMPLE_USERNAME_ENV')
passWord = os.getenv('EXAMPLE_PASSWORD_ENV')
```

Step 3 Specify **host** and **port**. It is advisable to store **host** and **port** in environment variables.

```
host = os.getenv('EXAMPLE_HOST_ENV')
port = os.getenv('EXAMPLE_PORT_ENV')
```

Step 4 Call psycopg2.connect(**kwargs) to create a database connection.

- In non-multi-primary mode, you need to specify the database name, server address, and port number.

  ```
  conn = psycopg2.connect(dbname="database", user=userName, password=passWord, host=host,
  port=port)
  ```

----End

5.6.2.2.3 Connection in SSL Mode

When connecting to GaussDB through psycopg2, you can enable SSL to encrypt client-server communication. This protects sensitive data during transmission.

Prerequisites

Log in to the GaussDB management console. On the **Instances** page, click an instance name to go to the **Basic Information** page. Click under **SSL** to download the root certificate or certificate bundle, and save the root certificate **ca.pem** to the client.

Procedure

Step 1 Import the **psycopg2**, **ssl**, and **os** modules.

```
import psycopg2
import ssl
import os
```

Step 2 Specify the username and password. It is advisable to store the username and password in environment variables because writing them directly in the code can create security risks.

```
userName = os.getenv('EXAMPLE_USERNAME_ENV')
passWord = os.getenv('EXAMPLE_PASSWORD_ENV')
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 3 Specify **host** and **port**. It is advisable to store **host** and **port** in environment variables.

```
host = os.getenv('EXAMPLE_HOST_ENV')
port = os.getenv('EXAMPLE_PORT_ENV')
```

Step 4 Set **sslmode** to **require**, **verify-ca**, or **verify-full** to enable SSL encryption. (Select the option that best fits your scenario.) For details, see **Table 5-109**.

```
# Set sslmode to require.
sslmode = "require"
# Set sslmode to verify-ca.
sslmode = "verify-ca"
# Set sslmode to verify-full.
sslmode = "verify-full"
```

Table 5-109 sslmode options

sslmode	Enable SSL Encryption	Description
disable	No	Try to set up a non-SSL connection.
allow	Possible	Try to set up a non-SSL connection first, and then an SSL connection if the trial fails.
prefer	Possible	Try to set up an SSL connection first, and then a non-SSL connection if the trial fails.
require	Yes	Try to set up an SSL connection. If there is a CA file, perform the verification according to the scenario in which the parameter is set to **verify-ca**.
verify-ca	Yes	Try to set up an SSL connection and check whether the server certificate is issued by a trusted CA.
verify-full	Yes	Try to set up an SSL connection, and check whether the server certificate is issued by a trusted CA and whether the host name of the server is the same as that in the certificate.

Step 5 Create a database connection in SSL mode.

- When calling psycopg2.connect(**kwargs) in non-multi-primary mode, you need to specify the database name, server address, and port number, and configure the root certificate **ca.pem**.

  ```
  conn = psycopg2.connect(dbname="database", user=userName, password=passWord, host=host, port=port, sslmode=sslmode, sslrootcert="ca.pem")
  ```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- In SSL mode, you are advised to use a Python interpreter that has been compiled by generating a DLL file (.so). To check the compilation mode of your Python interpreter, follow these steps:
  1. Run the **import ssl** command in the Python interpreter to import SSL.
  2. Run the **ps ux** command to query the PID of the Python interpreter. Assume that the PID is ********.
  3. In the shell CLI, run the **pmap -p** ******** **| grep ssl** command and check whether the command output contains the path related to **libssl.so**. If it does, the Python interpreter is compiled in dynamic link mode.
- If the Python interpreter is not compiled by generating a DLL file (.so), obtain **GaussDB-Kernel**_*Database version number*_*OS information*_**64bit_Libpq_Static.tar.gz**, decompress it, and copy **libpq.so.5** from the **lib** directory to replace the existing **libpq.so.5** in the **lib** directory of Python.

5.6.2.2.4 Connection Parameter Reference

Table 5-110 describes the commonly used connection parameters.

Table 5-110 Connection parameters

Parameter	Description	Parameter value.
dbname	Database name.	Default value: current OS username.
user	Username.	Default value: current OS username.
password	Password.	Default value: **NULL**.
host	Host name or IP address of the database to connect to. In non-multi-primary mode, you can specify multiple host names or IP addresses, separated by commas (,). They have a default type of Unix socket.	Default value: **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
port	Port number of the connection. • In non-multi-primary mode, this parameter indicates the port number of the host server. If multiple IP addresses have been specified as **host** and they use the same port number, enter that port number here. If these IP addresses use different port numbers, list the port numbers corresponding to each IP address and separate them with commas (,).	Default value: **5432**.
sslmode	SSL authentication mode.	Value range: **disable**, **allow**, **prefer**, **require**, **verify-ca**, and **verify-full**. For details about the value meanings, see **Table 5-109**. Default value: **prefer**.
sslcert	Path to the client certificate in the case of an SSL connection.	Default value: **NULL** (indicating that the client certificate file in the user's root directory is read).
sslkey	Path to the client key in the case of an SSL connection.	Default value: **NULL** (indicating that the key file in the user's root directory is read).
sslrootcert	Path to the root certificate in the case of an SSL connection.	Default value: **NULL** (indicating that the SSL root certificate in the user's root directory is read).
sslcompression	Specifies whether to compress the data sent via an SSL connection. Ignore this parameter when a non-SSL connection is in use.	Value range: • **0**: The data sent over an SSL connection is not compressed. • **1**: The data sent over an SSL connection is compressed. Default value: **1**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
sslcrl	Path to the SSL certificate revocation list (CRL). If the server certificate is present in the configured CRL, the client will reject authentication and close the connection during the SSL handshake phase.	Default value: **NULL**.
hostaddr	IP address of the database. Specify an IP address rather than a host name in this parameter, because host names cannot be parsed in this scenario. **NOTICE** • If both **host** and **hostaddr** are specified, **hostaddr** will take precedence.	Default value: **NULL**.
connect_timeout	Maximum timeout period of connections. You are advised not to set the connection timeout period to a value less than 2 seconds.	Unit: s. Default value: **0**, indicating no timeout limit.
client_encoding	Encoding format of the client.	Default value: client-side encoding format returned by the server during connection establishment.
application_name	Name of the application that connects to the database.	Default value: **NULL**.
fallback_application_name	Standby name of the application that connects to the database. If **application_name** is not explicitly set, **fallback_application_name** will automatically serve as the default application name of the connection.	Default value: **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
keepalives	Specifies whether to enable the TCP keepalive mechanism for the client and connection. The TCP keepalive mechanism is used to verify the validity of a connection that has been inactive for an extended period. It is commonly employed in IP-based network communication. **NOTICE** • The TCP keepalive mechanism periodically sends TCP probe packets to check the connection status. Keepalive parameters include **keepalives_idle**, **keepalives_interval**, and **keepalives_count**. 1. Idle time threshold: When the connection remains idle for longer than **keepalives_idle** (in seconds), the system will send the initial keepalive probe packet. 2. Subsequent probes: If no response is received, the system will send additional probe packets at **keepalives_interval**, with a maximum attempt limit of **keepalives_count**. 3. Connection status: If all probe attempts fail, the connection will be marked as down, and psycopg will throw an exception or try to reconnect. • In local inter-process communication scenarios such as those using a UDS, the TCP keepalive mechanism is unnecessary because there are no network delay or connection interruption issues. Therefore, if the connection is established through a UDS, you can ignore keepalive parameters.	Value range: • **1**: enabled. • **0**: disabled. Default value: **1**.
keepalives_idle	Time to wait before sending the initial TCP keepalive probe packet once the connection is identified as idle. In simple terms, if the database connection has been idle (since the last data transmission) for longer than **keepalives_idle**, the client will send TCP keepalive probe packets to the server.	Unit: s. Default value: **0**, indicating that the system TCP configuration is used.
keepalives_interval	Interval for the client to resend a probe packet if it does not receive an ACK response from the server.	Unit: s. Default value: **0**, indicating that the system TCP configuration is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
keepalives_count	Number of times that TCP keepalive probe packets are sent. It represents the maximum number of consecutive probe failures allowed. If this limit is exceeded, the connection will be marked as down.	Default value: **0**, indicating that the system TCP configuration is used.
options	Command-line options sent to the server when the connection starts. For example, **-c geqo=off** means setting the **geqo** parameter of the session to **off**.	Default value: **NULL**.
replication	Specifies whether the connection should use the replication protocol instead of common protocols. This protocol is designed for PG replication connections and internal tools such as pg_basebackup, but it can also be used by third-party applications.	Value range (the parameter values are case-insensitive): ● **true**, **on**, **yes**, **1**, and **reply**: used for intra-cluster connections in physical replication mode. ● **standby_cluster**, **hadr_main_standby**, and **hadr_standby_cn**: used for inter-cluster connections in physical replication mode. ● **database**: used for connections to the database specified by **dbname** in logical replication mode. ● **false**, **off**, **no**, **0**, and **NULL** (default value): used for regular connections, which are the default behavior. Default value: **NULL**. **NOTICE** In physical or logical replication mode, only simple query protocols can be used.
requirepeer	OS username of the server.	Default value: **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
target_session_attrs	Specifies the type of the host to be connected. A connection can be successfully established only when the actual server type matches the value of this parameter. This parameter takes effect only when multiple IP addresses are specified.	Value range: • **any**: The connection is applicable to all types of hosts. • **read-write**: The connection is limited to hosts that are both readable and writable. • **read-only**: The connection is limited to read-only hosts. • **primary**: The connection is limited to the primary node in the primary/standby system. • **standby**: The connection is limited to the standby node in the primary/standby system. • **prefer-standby**: The system first attempts to connect to the standby node in the primary/standby system. The **any** mode is used if the connection to all standby nodes on the **hosts** list fails. Default value: **primary**.
tcp_user_timeout	Maximum duration for data to remain unacknowledged before the kernel forcibly closes the TCP connection. This parameter is available on OSs that support the **tcp_user_timeout** socket option. This parameter does not take effect if Unix domain sockets are used to establish connections.	Unit: s. Default value: **0**, indicating that the system TCP configuration is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Parameter value.
rw_timeout	Sets the read and write timeout interval of the client connection. If a timeout occurs on the libpq side and the connection is closed, the running services delivered by libpq to the database will be forcibly terminated. This capability is controlled by the GUC parameter **check_disconnect_query**. It is supported when the parameter is set to **on** and not supported when the parameter is set to **off**.	Unit: s. Default value: **0**, indicating no timeout limit.

NOTE

- Psycopg2 builds its connection capabilities on the underlying libpq library but differs in load balancing capabilities as follows:
  - In non-multi-primary mode, since libpq does not support load balancing, psycopg2 also lacks load balancing capabilities.

5.6.2.3 Executing SQL Statements

NOTE

In Psycopg2, a transaction is started by default when a cursor object is created in the database. All SQL statements executed under the cursor are in the transaction. The connection object processes the transaction commit and rollback.

An application executes SQL statements to operate on the database, such as creating tables, inserting data, updating data, deleting data, and querying data. The **execute()** method is used to execute a single SQL statement, and the **executemany()** method is used to execute multiple SQL statements.

The following provides a brief example of using the execute() and executemany() methods (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example). (Prerequisites: You have connected to the database using the connection object **conn**.)

Step 1 Call the **cursor()** method to create a cursor object.

```
cur=conn.cursor()
```

Step 2 Call the execute() method to create the **student** table.

```
cur.execute("CREATE TABLE student(id integer,name varchar,gender varchar)")
```

Step 3 Call the execute() method to insert data one by one.

```
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(1,'Aspirin','M'))   # During execution, the placeholders (%s) will be automatically replaced by the actual parameter values.
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(3,'Dixheral','M'))
```

Step 4 Call the executemany() method to insert data in batches.

```
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",stus)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 5 Call the execute() method to update the student name with **id=1** to **'Aspirin Updated'**.

```
cur.execute("UPDATE student SET name=%s WHERE id=%s", ('Aspirin Updated', 1))
```

Step 6 Call the execute() method to delete the student record with **id=2**.

```
cur.execute("DELETE FROM student WHERE id=%s", (2,))
```

Step 7 Call the execute() method to query the result set.

```
cur.execute("SELECT * FROM student")
```

----End

5.6.2.4 Processing the Result Set

After executing a SELECT statement, you will need to obtain data from the result set. You have three choices: use **fetchone()** to obtain a single row of data, use **fetchmany(size)** to obtain a specified number of rows (specified by **size**), or use **fetchall()** to obtain all rows in the result. Below is a brief example (refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results** for the complete example).

After **calling the execute() method to query the result set**, use the fetchall() method to obtain all rows in the result.

```
results=cur.fetchall()
print (results)
```

NOTE

If the query result contains columns of a database-specific data type such as tinyint, these columns will be displayed in character string format.

5.6.2.5 Closing the Database Connection

After completing database operations, close the cursor by calling the **cur.close()** method and close the database connection by calling the **conn.close()** method.

Below is a brief example, assuming that the connection object is **conn** and cursor object is **cur**. (For the complete example, refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.)

Step 1 Call the cur.close() method to close the cursor.

```
cur.close()
```

Step 2 Call the conn.close() method to close the database connection.

```
conn.close()
```

----End

NOTICE

This method closes the database connection without automatically calling the commit() method to commit the current transaction. To permanently save the current transaction operations (such as INSERT, UPDATE, and DELETE) in the database, call the commit() method to commit the current transaction before calling close() to close the database connection.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.6.3 Typical Application Development Examples

5.6.3.1 Establishing a Database Connection, Executing SQL Statements, and Returning Results

Behavioral Results in Non-Multi-Primary Mode

This example demonstrates how to connect to a database, create a table, insert data, and return results with the Python driver.

```
# This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your environment.
# You need to provide the actual database name in dbname="database" and database port number in port=. host="localhost" indicates that the database is running on the localhost.
# To obtain certificates and other files, log in to the GaussDB management console. On the Instances page, click an instance name to go to the Basic Information page. Click the download button under SSL to download the root certificate or certificate bundle, and save the root certificate ca.pem to the client.

import psycopg2
import os
import ssl

# Obtain host and port from environment variables.
host = "localhost"
port = os.getenv('EXAMPLE_PORT_ENV')

# Obtain the username and password from environment variables.
userName = os.getenv('EXAMPLE_USERNAME_ENV')
passWord = os.getenv('EXAMPLE_PASSWORD_ENV')

# Connect to the database in non-encrypted mode.
conn = psycopg2.connect(dbname="database", user=userName, password=passWord, host=host, port=port)


# To establish an SSL connection to the database, use this line of code:
# conn = psycopg2.connect(dbname="database", user=userName, password=passWord, host=host, port=port,sslmode="verify-ca", sslrootcert="ca.pem")

# Create a cursor object.
cur=conn.cursor()
# Create a table.
cur.execute("CREATE TABLE student(id integer,name varchar,gender varchar)")

# Insert data.
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

# Insert data in batches.
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",stus)

# Update data.
cur.execute("UPDATE student SET name=%s WHERE id=%s", ('Aspirin Updated', 1))

# Delete data.
cur.execute("DELETE FROM student WHERE id=%s", (2,))

# Obtain the result.
cur.execute('SELECT * FROM student')
results=cur.fetchall()
print (results)

# Commit.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
conn.commit()

# Insert a data record.
cur.execute("INSERT INTO student(id,name,gender) VALUES(%s,%s,%s)",(7,'Lucy','F'))

# Roll back.
conn.rollback()

# Close the cursor and connection.
cur.close()
conn.close()
```

This example shows the following result:

```
[(1, 'Aspirin Updated', 'M'), (3, 'Dixheral', 'M'), (4, 'John', 'M'), (5, 'Alice', 'F'), (6, 'Peter', 'M')]
```

5.6.3.2 Using Logs

This example demonstrates how to use logs with the Python driver.

```
# This example assumes the username and password are stored in environment variables. Before running this example, set the environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in your environment.
# You need to provide the actual "database" and $port. "localhost" indicates that the database runs on the localhost.

import logging
import psycopg2
from psycopg2.extras import LoggingConnection
import os

# Obtain the username and password from environment variables.
userName = os.getenv('EXAMPLE_USERNAME_ENV')
passWord = os.getenv('EXAMPLE_PASSWORD_ENV')

logging.basicConfig(level=logging.DEBUG) # Log level.
logger = logging.getLogger(__name__)

db_settings = {
    "user": userName,
    "password": passWord,
    "host": "localhost",
    "dbname": "database",
    "port": $port
}

# By default, LoggingConnection logs all SQL statements. You can leverage the filter function to exclude unwanted or sensitive SQL statements. For example, exclude password-related SQL statements as follows:
class SelfLoggingConnection(LoggingConnection):

    def filter(self, msg, curs):
        if db_settings['password'] in msg.decode():
            return b'queries containing the password will not be recorded'
        return msg

# Create a connection object.
conn = psycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
cur = conn.cursor() # Create a cursor object.
cur.execute("select 1;")

# Close the cursor and connection.
cur.close()
conn.close()
```

This example shows the following result:

```
DEBUG:__main__:b'select 1;'
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- By default, LoggingConnection logs all SQL statements without anonymizing sensitive information. You can leverage the filter function to specify the output SQL content.
- Psycopg2 offers logging as an additional facility for developers to explicitly debug full SQL statements. This facility is not necessary by default. When activated, it will print SQL statements in debug logs before pyscopg2 executes the statements. It is important to note that this facility is not essential by default and should only be used when specific requirements dictate. For more information, visit https://www.psycopg.org/docs/extras.html?highlight=loggingconnection.

5.6.4 Psycopg API Reference

5.6.4.1 psycopg2.connect()

Description

Creates a database connection and returns the connection object.

Prototype

```
connect(dsn, connection_factory, cursor_factory, **kwargs)
```

Parameters

Table 5-111 psycopg2.connect parameters

Keyword	Description
dsn	Specifies the data source name in string format. It includes essential information required for a database connection, such as the database name, server address, port number, username, and password. The default value is **None**.
connection_factory	Provides a simple method for creating custom cursors. The default value is **None**.
cursor_factory	Uses a custom cursor. The default value is **None**.
**kwargs	Passes a variable number of keyword arguments to the connection function. Commonly used connection parameters are explained in **Connection Parameter Reference**.

Return Values

Connection object (for connecting to a database instance).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.2 connection.cursor()

Description

Creates a database cursor object to execute SQL statements, manage query results, and control transactions in the database.

Prototype

```
cursor(name, cursor_factory, scrollable, withhold)
```

Parameters

Table 5-112 connection.cursor parameters

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

Return Values

Cursor object.

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.3 cursor.execute(query,vars_list)

Description

Executes a parameterized SQL statement. The psycopg2 module supports using **%s** as placeholders.

Prototype

```
cursor.execute(query,vars_list)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-113 cursor.execute parameters

Keyword	Description
query	SQL statement to be executed.
vars_list	List of variables used to replace placeholders (%s) in the SQL statement.

Return Values

None

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.4 cursor.executemany(query,vars_list)

Description

Executes the same SQL statement multiple times, which is especially useful for batch data insertion.

Prototype

```
cursor.executemany(query,vars_list)
```

Parameters

Table 5-114 cursor.executemany parameters

Keyword	Description
query	SQL statement to be executed.
vars_list	List of tuples or dictionaries. Each tuple or dictionary corresponds to an SQL statement, and the elements they contain will replace the placeholders (%s) in the SQL statement.

Return Values

None

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.5 connection.commit()

Description

Commits the current transaction, which permanently saves all pending database operations (such as INSERT, UPDATE, and DELETE) in the database and ends the transaction. Specifically:

- Persistent change: The connection will permanently write all data modifications made since the last commit or rollback into the database.
- Transaction ending: The current transaction will end upon a commit. Subsequent operations will belong to a new transaction, and the new transaction will take effect only after another commit.

NOTICE

By default, psycopg starts a new transaction before executing the first command. If commit() is not called, the transaction will automatically roll back when the connection is closed, resulting in the loss of all data modifications.

Prototype

```
connection.commit()
```

Parameters

None

Return Values

None

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.6 connection.rollback()

Description

Rolls back the current transaction, which reverts all uncommitted modifications to maintain data consistency.

Prototype

```
connection.rollback()
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

None

Return Values

None

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.7 cursor.fetchone()

Description

1. Obtains a single row of data: After an SQL query is executed, fetchone() returns the next row of data (in tuple format) from the result set. If no data has been read, the first call returns the first row.
2. Returns **None** if no data is found: If all rows have been read, subsequent calls will return **None**, indicating that the result set has been fully processed.
3. Controls the cursor: Each time fetchone() is called, the cursor advances by one row, allowing for sequential traversal of all results.

Prototype

```
cursor.fetchone()
```

Parameters

None

Return Values

A single tuple, which is the first result in the result set. If no data is available in the result set, **None** will be returned.

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.8 cursor.fetchmany(size)

Description

After an SQL query is executed, calling fetchmany(size) fetches the specified number of rows from the cursor's remaining unread results. Each call resumes reading from the point where the previous call left off, until the result set is fully exhausted.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Prototype

```
cursor.fetchmany(size)
```

Parameters

Table 5-115 cursor.fetchmany parameters

Keyword	Description
size	Integer, indicating the number of rows fetched each time.

Return Values

A maximum of **size** rows, though possibly fewer. For details, see **Table 5-116**.

Table 5-116 Return values in different scenarios

Scenario	Return
The number of remaining rows in the unread result set of the cursor is greater than or equal to **size**.	**size** rows are returned.
The number of remaining rows in the unread result set of the cursor is less than **size**.	All remaining rows are returned.
The unread result set of the cursor has no data.	An empty list is returned.

Examples

None

5.6.4.9 cursor.fetchall()

Description

Obtains all (remaining) rows in the query result and returns them as a list of tuples. This API is well-suited for small result sets.

- Scenario 1 - A query retrieves five records: Directly calling cursor.fetchall() returns a list of five tuples. Each tuple represents a row of data, with fields arranged in the order they were queried.
- Scenario 2 - A query retrieves five records and fetchone() has been called first: Because the fetchone() API has returned the first row of data in the result set and advanced the cursor by one row, calling cursor.fetchall() at this point will return only the remaining four rows of data as a list of tuples.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- Cursor status: Once fetchall() is called, the cursor will be placed at the end of the result set. Any further calls to fetchall() will return an empty list.
- Memory risk: When dealing with a large amount of data, exercise caution when using this API as it may cause memory overflow.

Prototype

```
cursor.fetchall()
```

Parameters

None

Return Values

A list of tuples, representing all results in the result set. If no data is available in the result set, an empty list will be returned.

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.6.4.10 cursor.close()

Description

Closes the cursor of the current connection.

Prototype

```
cursor.close()
```

Parameters

None

Return Values

None

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.6.4.11 connection.close()

Description

Closes the database connection.

NOTICE

This method does not call commit() automatically. It simply closes the database connection. Failing to call commit() before closing the database connection will lead to the loss of all changes made.

Prototype

```
connection.close()
```

Parameters

None

Return Values

None

Examples

Refer to **Establishing a Database Connection, Executing SQL Statements, and Returning Results**.

5.7 Development Based on the Go Driver

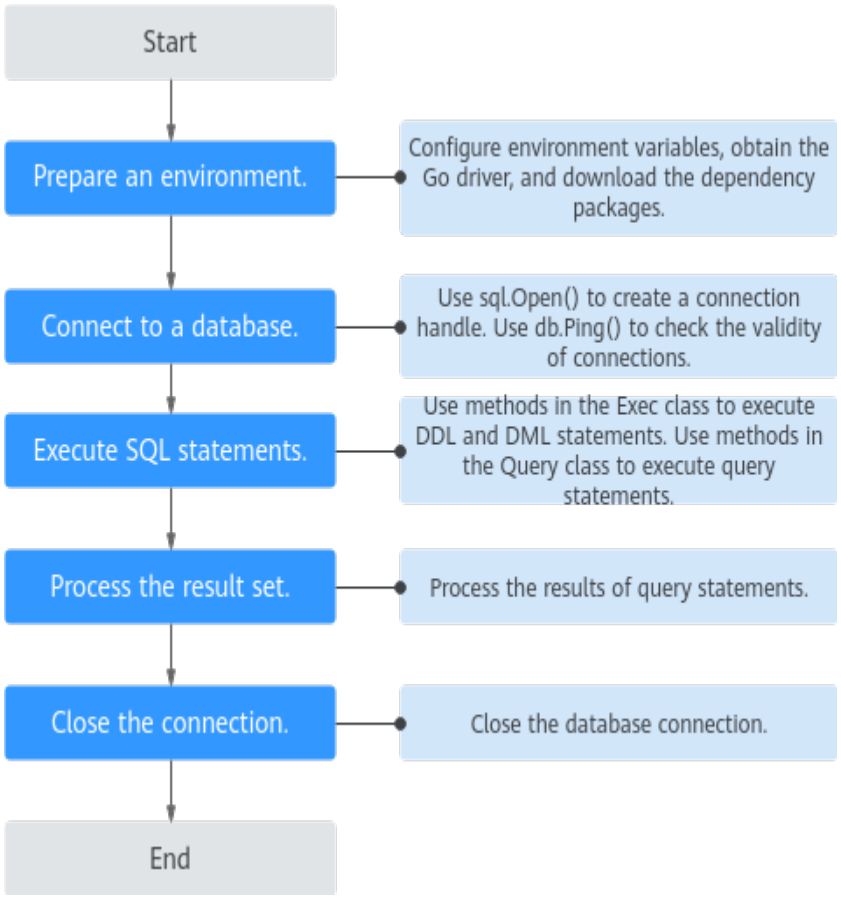
5.7.1 Development Process

Developing applications based on the Go driver depends on the native **database/sql** package and the driver in GaussDB. The **database/sql** package offers standard APIs, while GaussDB implements the APIs to be called by the **database/sql** package.

The application development process based on Go (see **Figure 5-19**) involves these steps: prepare an environment, connect to a database, execute SQL statements, process the result set, and close the connection.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-19 Application development process based on Go



5.7.2 Development Procedure

5.7.2.1 Preparing an Environment

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 5-117**.

Table 5-117 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

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

- Verifying the software package integrity on a Linux server:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

a. Upload the software package and verification package to the same directory on the VM.

b. Run the following command to verify the integrity of the software package:

```
F=$(echo DBS-GaussDB-driver*.tar.gz);echo "$(awk '{print $1}' "$F.sha256")  $F" | sha256sum -c
-
```

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

```
DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz: OK
```

- Verifying the software package integrity on a Windows server:

a. Press **Win+R** to open the **Run** dialog box. Type **cmd** in the text box and press **Enter** to open the **Command Prompt** window.

b. Run the following command to obtain the hash value of the driver package:

certutil -hashfile *{local_directory_of_the_driver_package}**{driver_package_name}* **sha256**

- Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.
- Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz**.

Example: **certutil -hashfile** *C:\Users\DBS-GaussDB-driver_507.0_20260629130806.133858239.tar.gz* **sha256**

c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 5-117**.

- If they are consistent, the verification is successful.
- If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining the Go Driver Package

Step 1 Decompress the obtained driver package. The file hierarchy of the driver package is as follows: **DBS-GaussDB-driver_*/***Database deployment mode*/***Driver type*/***OS architecture*/***Release package*/***Driver package*.

Step 2 Select a database deployment mode.

The driver package is independent of the deployment mode. You can use the driver package for either of the deployment modes: Centralized or Distributed.

Step 3 Select a driver type.

Select **go_driver.tar.gz** and decompress it.

Step 4 Select an OS architecture.

1. Non-encrypted functionalities are independent of OS architectures, giving you the freedom to select any OS architecture.
2. Encrypted functionalities are specific to OS architectures. To determine your client OS architecture, run the following command:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
uname -a
```

Step 5 Select the Go release package.

Step 6 Decompress the Go release package to obtain the source code package of the Go driver: **openGauss-connector-go-pq**.

----End

Setting Environment Variables

- **GO111MODULE**: Set **GO111MODULE** to **on** when installing the Go driver by importing a file online.
  ```
  export GO111MODULE=on
  ```

  NOTE

  If you do not want to reconstruct the **go mod** project, set **GO111MODULE** to **off** and manually download the dependency packages. The dependency packages must be at the same level as the driver root directory and service code.

- **GOPROXY**: When importing data online, you need to configure the path *[path-to-goproxy]* that contains the Go driver package.
  ```
  export GOPROXY=[path-to-goproxy]
  ```
- **GOPATH**: specifies the path *[path-to-gopath]* for storing the Go driver dependency packages.
  ```
  export GOPATH=[path-to-gopath]
  ```
- The Go version must be 1.13 or later. You can run the following command to check the value of the Go environment variable *GOVERSION*:
  ```
  go env
  ```
- You can set other Go environment variables based on your needs.

Downloading Dependency Packages

Step 1 When **GO111MODULE** is set to **on** to enable Go modules, go to the root path of the Go driver code and run the following command to download the dependencies required by the driver code:

```
go mod tidy
```

Step 2 After **Step 2** is complete, add the following line to the **go.mod** file to replace the path of the module on which the driver package depends:

```
replace [Go driver module name] => [Local path of the Go driver package]
```

NOTE

- Change *[Go driver module name]* to the actual Go driver module name.
- Change *[Local path of the Go driver package]* to the actual local path of the Go driver.

The following uses the Go driver referenced in *xxx*.**go** as an example:

```
// xxx.go
package main
import (
  _ "openGauss-connector-go-pq"
)
```

Add the following content to the **go.mod** file:

```
replace openGauss-connector-go-pq => [path-to-driver]/openGauss-connector-go-pq
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- In the example *xxx*.**go**, **openGauss-connector-go-pq** indicates the driver module name. You can change it as required.
- In **go.mod**, *[path-to-driver]*/**openGauss-connector-go-pq** indicates the path to the local Go driver package. With this line, the Go driver module name **openGauss-connector-go-pq** referenced in *xxx*.**go** points to the local Go driver in *[path-to-driver]*/**openGauss-connector-go-pq**.

----End

NOTICE

- When you run the **go mod tidy** command to download dependencies, some of them may be downloaded as an earlier version. If the earlier version has vulnerabilities, you can change the dependency version in the **go.mod** file and update the dependency to the version after the vulnerability is fixed to avoid risks.
- You can use Go 1.13 or later. The runtime library needs to be updated to 1.18 or later.

5.7.2.2 Connecting to a Database

5.7.2.2.1 Connection in Non-encrypted Mode

When using the Go driver for development, set connection parameters to generate a connection string. Then call the sql.Open() API from the **database/sql** package of Golang to create an SQL database connection object. Note that sql.Open() does not immediately establish a database connection. The connection will only be established when the database is first accessed, such as when executing a query or utilizing db.Ping(). You can call db.Ping(), a standard Go SQL API, to check whether the database connection can be successfully established. This API initiates connection attempts when necessary, with the initial attempt occurring right after sql.Open() is called.

sql.Open() Function Prototype

The Go driver provides the following method to generate a database connection object:

```
func Open(driverName, dataSourceName string) (*DB, error)
```

Parameters:

- **driverName** indicates a driver name. The database driver options are as follows:
  - **"gaussdb"**: The driver name **"gaussdb"** is required for connecting to GaussDB. In this example, the database driver name is **"gaussdb"**.
  - For details about other Go driver names, see **Go Driver Compatibility**.
- **dataSourceName** indicates a connection string, which can be a DSN or URL.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- DSN format: key1 = value1 key2 = value2... Different groups of keywords are separated by space. The space on the left and right of the equal sign (=) is optional.
- URL format: driverName://[userspec@][hostspec][/dbname][?paramspec]

  driverName indicates a driver name, and **"gaussdb"** is used as the database driver name.

  userspec indicates user[:password]. When a URL is used for connection, the password cannot contain separators in the URL string. If the password contains separators, the DSN format is recommended. In addition, the initial user cannot be used for remote login.

  hostspec indicates [host][:port][, ...].

  dbname indicates the database name.

  paramspec indicates name=value [&...].

NOTICE

- In the DSN format, if there are multiple IP addresses:
  - When the value of **num(ip)** is the same as that of **num(port)**, the IP address matches the port number.
  - When the value of **num(ip)** is greater than that of **num(port)**, the IP address that cannot match the port number matches the first port number. For example, the matching condition of **host = ip1, ip2, ip3 port = port1, port2** is **ip1:port1, ip2:port2, ip3:port1**.
  - When the value of **num(ip)** is smaller than that of **num(port)**, the extra port numbers are deprecated. For example, the matching condition of **host = ip1, ip2, ip3 port = port1, port2, port3, port4** is **ip1:port1, ip2:port2, ip3:port3**.
- In the URL format, if there are multiple IP addresses:
  - In the URL, *ip*:*port* must appear in pairs, that is, the value of **num(ip)** is the same as that of **num(port)**. Use commas (,) to separate multiple pairs. Example: **gaussdb://user:password@ip1:port1, ip2:port2, ip3:port3/gaussdb**.
  - The URL contains only multiple IP addresses. The port number is specified by an environment variable or the default value **5432**. For example, in the case of **gaussdb://user:password@ip1, ip2, ip3/gaussdb**, ,if the environment variable *PGPORT* is set to **"port1, port2"**, the mapping is **ip1:port1, ip2:port2, ip3:port1**. If the environment variable is not set, the mapping is **ip1:5432,ip2:5432,ip3:5432**.
- When entering IPv6 addresses in the URL connection string, remember to enclose the IP addresses in square brackets ([]), for example, **gaussdb://user:password@[ip1]:port1, [ip2]:port2, [ip3]:port3/gaussdb**.

db.Ping () Function Prototype

The **database/sql** package of Golang provides the ping API bound to the database object to test whether the connection can be successfully established after the database object is created. The API definition is as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
func (db *DB) Ping() error
```

You can use the db.Ping() function to test whether the database is successfully connected. The following code can help you understand how to use the db.Ping() function:

```
err = db.Ping()
if err != nil {
   log.Fatal("Connection failed:", err)
   return
}
log.Println("Successfully connected to the database.")
```

If the value of **err** is **nil**, the connection has been successfully established without any error. In this scenario, the code above will show "Successfully connected to the database." If the value of **err** is not **nil**, the database connection fails and the error message is received. You can use log.Fatal to output the error information. For example, when a database connection fails, db.Ping() returns the error message "connection refused." In this case, **err** receives the error message. If log.Fatal("Connection failed:," err) is called, "Connection failed: connection refused" is displayed.

Method 1

Set connection parameters in the DSN or URL connection string. For example, set the IP address, port number, username, and password in the corresponding *[GOXX]* environment variables. Subsequently, the service code calls the os.Getenv() method to obtain the necessary variable values for concatenating the DSN or URL connection string. The process unfolds as follows:

Step 1 Set the environment variables required for connection string concatenation.

```
export GOHOSTIP="XXX.XXX.XXX.XXX"
export GOPORT=XXX
export GOUSRNAME="XXX"
export GOPASSWD="XXXXXXX"
```

Step 2 Import the Go driver module and other standard dependency libraries.

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
   "database/sql"
   _ [Go driver module name] // Replace it with the actual Go driver module name.
   "os"
   "log"
)
```

NOTE

- **database/sql**: Golang standard library, which provides common database operation APIs and defines core methods (such as Query(), Exec(), and Prepare()) for interacting with databases.
- _ **[***Go driver module name***]**: Go driver module.
- **os**: Golang standard library, which provides functions (file operations and environment variables) for interacting with the OS. In this example, the **os** package is used to obtain parameter values from environment variables. The driver does not depend on this package.
- **log**: Golang standard library, which provides the log recording function and is usually used to output error messages or debug logs. In this example, the **log** package is used to print logs. The driver does not depend on this package.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 3 Call the os.Getenv() method to obtain the IP address, port number, username, and password from environment variables.

```
hostip := os.Getenv("GOHOSTIP")
port := os.Getenv("GOPORT")
usrname := os.Getenv("GOUSRNAME")
passwd := os.Getenv("GOPASSWD")
```

Step 4 Define a DSN or URL connection string and call the sql.Open() function to create a database connection object.

```
str := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=gaussdb sslmode=disable" // DSN connection string. You need to set the values of dbname and
sslmode.
// str := "gaussdb://" + usrname + ":" + passwd + "@" + hostip + ":" + port + "/gaussdb?sslmode=disable" //
URL connection string. You need to set the values of the database name and sslmode.
db, err := sql.Open("gaussdb", str)  // Create a database connection object. The database connection is not
established immediately.
if err != nil {
    log.Fatal(err) // When an error is reported during execution, use the log.Fatal method to output an error
message.
}
```

Step 5 Call db.Ping() to check whether the database connection is successful.

```
err = db.Ping()
if err != nil {
    log.Fatal(err)  // If the value of err is not nil, the connection fails. In this case, use log.Fatal to output the
connection failure cause.
}
```

----End

NOTE

Steps 2 to 5 provide sample service code for reference.

Method 2

Certain connection parameters are set in *[PGXX]* environment variables rather than the DSN or URL connection string. For example, to set the username and password in the DSN or URL connection string, use the environment variables *PGHOST*, *PGPORT*, *PGDATABASE*, and *PGXX* to specify the IP address, port number, database name, and other parameters respectively. The procedure is as follows:

Step 1 Set environment variables.

```
export PGHOST="XXX.XXX.XXX.XXX"
export PGPORT=XXX
export PGDATABASE="XXX"
export PGSSLMODE="disable"
export GOUSRNAME="XXX"
export GOPASSWD="XXXXXXX"
```

Step 2 Import the Go driver module and other standard dependency libraries.

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
  "database/sql"
  _ [Go driver module name] // Replace it with the actual Go driver module name.
  "os"
  "log"
)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- **database/sql**: Golang standard library, which provides common database operation APIs and defines core methods (such as Query(), Exec(), and Prepare()) for interacting with databases.
- _ "*[Go driver module name]*": Go driver module. You can perform **this step** to point the Go driver module to the local Go driver path.
- **os**: Golang standard library, which provides functions (file operations and environment variables) for interacting with the OS. In this test case, the **os** package is used to obtain parameter values from environment variables. The driver does not depend on this package.
- **log**: Golang standard library, which provides the log recording function and is usually used to output error messages or debug logs. In this test case, the **log** package is used to print logs. The driver does not depend on this package.

Step 3 Call the os.Getenv() method to obtain the username and password.

```
usrname := os.Getenv("GOUSRNAME")
passwd := os.Getenv("GOPASSWD")
```

Step 4 Define a DSN or URL connection string and call the sql.Open() function to create a database connection object.

```
str := " user=" + usrname + " password=" + passwd // DSN connection string.
// str := "gaussdb://" + usrname + ":" + passwd  // URL connection string.
db, err := sql.Open("gaussdb", str)  // Create a database connection object. The database connection is not established immediately.
if err != nil {
    log.Fatal(err)  // When an error is reported during execution, use the log.Fatal method to output an error message.
}
```

Step 5 Call db.Ping() to check whether the database connection is successful.

```
err = db.Ping()
if err != nil {
    log.Fatal(err)  // If the value of err is not nil, the connection fails. In this case, use log.Fatal to output the connection failure cause.
}
```

----End

NOTE

Steps 2 to 5 provide sample service code for reference.

5.7.2.2.2 Connection in SSL Mode

When an application connects to GaussDB through the Go driver, you can enable SSL to encrypt client-server communication. When SSL encryption is enabled, the Go driver uses TLS 1.3 by default. The minimum standard protocol for supporting SSL encryption is TLS 1.2. This section describes how to configure a client in SSL mode.

Prerequisites

You have obtained the certificate and private key files required by the server and completed the server configuration. In addition, you have obtained the **client.crt** client certificate, **cacert.pem** root certificate, and **client.key** client private key file required by the client.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> For instructions on how to generate and obtain certificates, contact an administrator. For instructions on how to configure certificates on the server, contact an administrator.

Using the require and verify-ca Verification to Connect to a Database

Step 1 Set environment variables.

```
export GOHOSTIP="XXX.XXX.XXX.XXX"
export GOPORT=XXX
export GOUSRNAME="XXX"
export GOPASSWD="XXXXXXX"
export GOSSLPASSWD="XXXXXXX"
```

Step 2 Import the dependency package required by the connection.

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
   "database/sql"
   _ [Go driver module name] // Replace it with the actual Go driver module name.
   "os"
   "log"
)
```

Step 3 Call the os.Getenv() method to obtain the IP address, port number, username, password, and SSL passphrase from the environment variables.

```
hostip := os.Getenv("GOHOSTIP")
port := os.Getenv("GOPORT")
usrname := os.Getenv("GOUSRNAME")
passwd := os.Getenv("GOPASSWD")
sslpasswd := os.Getenv("GOSSLPASSWD")
```

Step 4 Define the connection string in DSN format.

```
dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=gaussdb sslcert=certs/client.crt sslkey=certs/client.key sslpassword=" + sslpasswd
```

Step 5 Configure **sslmode**.

```
parameters := []string{
  " sslmode=require",
  " sslmode=verify-ca sslrootcert=certs/cacert.pem",
}
```

Step 6 Enable SSL encryption and use the **require** and **verify-ca** verification to connect to the database.

```
for _, param := range parameters {
  db, err := sql.Open("gaussdb", dsnStr+param)
  if err != nil {
   log.Fatal(err)
  }
   err = db.Ping()  // Check whether the database is successfully connected.
   if err != nil {
      log.Fatal(err)
   }
}
```

----End

> **NOTE**
>
> Steps 2 to 6 provide sample service code for reference.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Using the RSA and ECDSA Certificates to Connect to a Database

Step 1 Set environment variables.

```
export GOHOSTIP="XXX.XXX.XXX.XXX"
export GOPORT=XXX
export GOUSRNAME="XXX"
export GOPASSWD="XXXXXXX"
export GOSSLPASSWD="XXXXXXX"
```

Step 2 Import the dependency package required by the connection.

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
   "database/sql"
   _ [Go driver module name] // Replace it with the actual Go driver module name.
   "os"
   "log"
)
```

Step 3 Call the os.Getenv() method to obtain the IP address, port number, username, password, and SSL passphrase from the environment variables.

```
hostip := os.Getenv("GOHOSTIP")
port := os.Getenv("GOPORT")
usrname := os.Getenv("GOUSRNAME")
passwd := os.Getenv("GOPASSWD")
sslpasswd := os.Getenv("GOSSLPASSWD")
```

Step 4 Define the connection string in DSN format.

```
dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=gaussdb sslcert=certs/client.crt sslkey=certs/client.key sslpassword=" + sslpasswd
```

Step 5 Set SSL-related parameters.

```
connStrs := []string{
  " sslmode=verify-ca sslcert=certs/client_rsa.crt sslkey=certs/client_rsa.key sslpassword=" + sslpasswd + "
sslrootcert=certs/cacert_rsa.pem",
  " sslmode=verify-ca sslcert=certs/client_ecdsa.crt sslkey=certs/client_ecdsa.key sslpassword=" + sslpasswd +
" sslrootcert=certs/cacert_ecdsa.pem",
}
```

Step 6 Use the RSA and ECDSA certificates to connect to the database.

```
for _, connStr := range connStrs {
  db, err := sql.Open("gaussdb", dsnStr+connStr)
  if err != nil {
   log.Fatal(err)
  }
   err = db.Ping()  // Check whether the database is successfully connected.
   if err != nil {
      log.Fatal(err)
   }
}
```

----End

NOTE

Steps 2 to 6 provide sample service code for reference.

5.7.2.2.3 Connection Parameter Reference

Common Connection Parameters

Table 5-118 lists the common connection parameters of the Go driver.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-118 Database connection parameters

Parameter	Description
host	IP address of the host. By default, the localhost is used. You can also specify it using the environment variable *PGHOST*.
port	Port number of the host. The default value is **5432**. You can also specify it using the environment variable *PGPORT*.
dbname	Database name, which can also be specified by the environment variable *PGDATABASE*.
user	Username, which can also be specified by the environment variable *PGUSER*.
password	Password.
connect_timeout	Timeout period for connecting to the server. It can also be specified using the environment variable *PGCONNECT_TIMEOUT*. The default value is **0**, indicating that no timeout occurs.
sslmode	SSL encryption mod. It can also be specified using the environment variable *PGSSLMODE*. The default value is **prefer**. Value range: • **disable**: SSL connections are not used. • **allow**: SSL connections are used when required by the database server. However, this mode does not verify the authenticity of the database server. • **prefer**: SSL connections are preferred if the database supports SSL. However, this mode does not verify the authenticity of the database server. • **require**: SSL connections are required. This mode only encrypts data but does not verify the authenticity of the database server. • **verify-ca**: SSL connections are required. The server must present a certificate issued by a trusted CA. • **verify-full**: SSL connections are required. The server must present a certificate issued by a trusted CA, and the server's host name must match the name in the certificate.
sslkey	Path to the client key in the case of an SSL connection. It can also be specified using the environment variable *PGSSLKEY*.
sslcert	Path to the client certificate in the case of an SSL connection. It can also be specified using the environment variable *PGSSLCERT*.
sslrootcert	Path to the root certificate in the case of an SSL connection. It can also be specified using the environment variable *PGSSLROOTCERT*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
sslcrl	Path to the SSL certificate revocation list (CRL). It can also be specified using the environment variable *PGSSLCRL*. If the server certificate is present in the configured CRL, the client will reject authentication and close the connection during the SSL handshake phase.
sslpassword	Password used to decrypt the SSL key file. If this parameter is enabled, the SSL key file is encrypted for storage. Currently, the SSL key file can be encrypted using DES or AES. **NOTE** The DES encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.
disable_prepared_binary_result	Specifies whether a connection uses the binary format when receiving query results from a prepared statement. This parameter is a string and is used only for debugging. The default value is **no**. Value range: ● **yes**: The connection does not use the binary format when receiving query results from a prepared statement. ● **no**: The connection uses the binary format when receiving query results from a prepared statement.
binary_parameters	Specifies whether to send []byte in the binary format during connection. This parameter is a character string. The default value is **no**. Value range: ● **yes**: The []byte is sent in the binary format. The recommended value is **yes**. Binding parameters as []byte can reduce internal type conversion. ● **no**: The []byte is not sent in the binary format.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
target_session_attrs	Connection type of the database, which can also be specified by the environment variable *PGTARGETSESSIONATTRS*. This parameter is used to identify the primary and standby nodes. There are six value options, namely, **any** (default value), **master**, **slave**, **preferSlave**, **read-write**, and **read-only**. • **any**: attempts to connect to any DN in the URL connection string. • **master**: attempts to connect to a primary node in the URL connection string. If the primary node cannot be found, an exception is thrown. • **slave**: attempts to connect to a standby node in the URL connection string. If the standby node cannot be found, an exception is thrown. • **preferSlave**: attempts to connect to the standby DN specified in the URL connection string. If no standby DN is available, the connection will be made to the primary DN. • **read-write**: The connection is limited to hosts that are both readable and writable. • **read-only**: The connection is limited to read-only hosts.
loggerLevel	Log level, determining which logs will be printed. It can also be specified using the environment variable *PGLOGGERLEVEL*. The value can be **trace**, **debug**, **info**, **warn**, **error**, or **none**, in descending order of priority. The default value is **error**. **NOTE** If log or error information contains security-related keywords (such as PASSWORD, GS_ENCRYPT, and IDENTIFIED), the Go driver uses ****** to replace related information.
application_name	Name of the application that connects to the database. The default value is **go-driver**. You are advised not to configure this parameter.
RuntimeParams	Value of a GUC parameter of the USERSET type that is run by default when a session is connected. For example, parameters such as **search_path**, **application_name**, and **statement_timeout** can be configured in the connection string.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description

autoBalance	Load balancing policy in a distributed environment. The value is a character string. The value can be **true**, **balance**, **roundrobin**, **shuffle**, **priority***n*, or **false** (default value).
	• **true**, **balance**, or **roundrobin**: SQL load balancing is enabled for the Go driver to balance multiple connections from an application to available CNs in the database cluster.
	Example:
	`gaussdb://user:password@host1:port1,host2:port2/database?autoBalance=true`
	The driver periodically updates the available CN list in the cluster. The update interval is specified by the **recheckTime** parameter. The default value is **10s**. Here is an example list:
	`host1:port1,host2:port2,host3:port3,host4:port4`
	When load balancing is enabled, **host1** and **host2** ensure high availability only when they are connected for the first time. Then, the driver selects available CNs from **host3**, **host4**, **host1**, and **host2** in sequence to update the available CN list. The Connector.Connect called by the user uses the RoundRobin algorithm to select CNs from **host1**, **host2**, **host3**, and **host4** for connection.
	• **priority***n*: Driver priority-based load balancing is enabled to balance multiple connections from an application to the first *n* available CNs configured in the URL. If all *n* CNs are unavailable, connections will be randomly allocated to other available CNs in the database cluster. *n* is a positive integer and is less than the number of CNs configured in the URL.
	Example:
	`gaussdb://user:password@host1:port1,host2:port2,host3:port3/database?autoBalance=priority2`
	The driver periodically obtains the list of available CNs in the entire cluster. Here is an example list:
	`host1:port1,host2:port2,host3:port3,host4:port4,host5:port5,host6:port6`
	host1 and **host2** are in AZ 1, and **host3** and **host4** are in AZ 2. The driver preferentially selects **host1** and **host2** for load balancing. If both **host1** and **host2** are unavailable, the driver randomly selects CNs from **host3**, **host4**, **host5**, and **host6** for connection.
	• **shuffle**: Driver random load balancing is enabled to randomly balance multiple connections from an application to available CNs in the database cluster.
	Example:
	`gaussdb://user:password@host1:port1,host2:port2,host3:port3/database?autoBalance=shuffle`
	The driver periodically obtains the list of available CNs in the entire cluster. Here is an example list:
	`host1:port1,host2:port2,host3:port3,host4:port4`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
	host1, **host2**, and **host3** ensure high availability only when they are connected for the first time. Then, they use the shuffle algorithm to randomly select a CN from the updated CN list for connection. • **false**: Driver load balancing and priority-based load balancing are disabled. **NOTE** • Load balancing is based on the connection level rather than the transaction level. If connections are persistent and the load on the connections is unbalanced, the load on the CNs may be unbalanced. • Load balancing can be used only in distributed scenarios.
recheckTime	Specifies the interval at which the system checks the CN status in the database cluster and obtains the IP address list of available CNs. The value is an integer ranging from 5 to 60, in seconds. The default value is **10s**.
usingEip	Specifies whether to use service IP addresses for load balancing. This parameter is a character string. The default value is **yes**. • **yes**: Service IP addresses (corresponding to the **node_host1** and **node_port1** columns in the pgxc_node system catalog) are used for load balancing. • **no**: Data IP addresses (corresponding to the **node_host** and **node_port** columns in the pgxc_node system catalog) are used for load balancing.
tcp_syn_retries	Specifies the maximum number of retries during TCP connection establishment. If this parameter is not set, the system uses the default number of retries. You are advised to use the following formula to ensure proper configuration: Number of retries x Retry interval < **connect_timeout**. **NOTE** The retry interval is implemented at the bottom layer of the system and does not need to be manually configured.
socketTimeout	Sets the read/write timeout interval. If the execution of a service statement or the reading of data streams from the network exceeds the timeout interval, the connection is interrupted. **NOTE** This parameter specifies the maximum execution time of a single SQL statement. If the execution time of a single SQL statement exceeds this value, the statement is interrupted and reconnected. You are advised to set this parameter based on service characteristics. If this parameter is not set, the default value **0** is used, indicating that no timeout occurs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
timeZone	After this parameter is set, the database GUC parameter **TimeZone** is set to the value of this parameter during connection establishment. When you query timestamp and date data without time zones, the time zone specified by this parameter is automatically added to the returned result. This parameter is a character string. Value range: same as the value range of the database GUC parameter **TimeZone**. For details about the value range of the GUC parameter **TimeZone**, see "Database GUC Parameters > GUC Parameters > Default Settings for Client Connections > Locale and Formatting" in *Reference*.

Priority of Connection Parameter Values

The Go driver can obtain connection parameter values from the following sources:

- DSN/URL connection string (explicitly specified)
- *[PGXX]* environment variables (certain parameters can be specified in environment variables, as described in **Table 5-118**)
- Default values

The priority sequence for connection parameter values obtained from these sources is: DSN/URL connection string > *[PGXX]* environment variables > Default values. The following uses a port number in a connection parameter as an example:

1. When a database connection object is created, a port number is preferentially obtained from a DSN or URL connection string. If the **port** parameter is configured in the connection string, the value is used as the port number to create a database connection object.
2. If the **port** parameter is not configured in the connection string, the Go driver further checks whether the environment variable *PGPORT* is configured. If the value of the environment variable *PGPORT* is not empty, the value is used as the port number to create a database connection object.
3. If neither the connection string nor the environment variable is configured, the default port number **5432** is used to create a database connection object.

NOTE

- *[PGXX]* is an environment variable name. For actual names, refer to parameter descriptions in **Table 5-118**.
- Before using values from *[PGXX]* environment variables to create a database connection object, it is advisable to confirm that the values are the expected ones.
- Environment variables can be configured by running the **export** *PGXX*="*value*" command. Replace *[PGXX]* with an environment variable in **Table 5-118**, and replace *value* with its configured value. The following uses *PGTARGETSESSIONATTRS* as an example:
  ```
  export PGTARGETSESSIONATTRS="master"
  ```
- For detailed configuration of connection parameters, refer to **Method 1** or **Method 2**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Initial Packet During Connection Establishment

After a socket is established during the connection process between the driver and the database, the first packet sent by the driver to the database, in key-value format, is called the initial packet. This packet includes the username (**user**), database name (**database**), and certain GUC parameters (**enable_full_encryption**, **application_name**, and **RuntimeParams**). **RuntimeParams** defines the USERSET-type GUC parameters supported by the database, not parameter names. For details, see **RuntimeParams**. For detailed descriptions of these GUC parameters, refer to "Database GUC Parameters > GUC Parameters" in *Reference*.

NOTICE

After the database executes the following command, the values of the preceding GUC parameters in the initial packet will be updated to the values specified in the initial packet:

```
reset all;
```

5.7.2.3 Executing SQL Statements

The Exec class methods in **type DB** are used to execute DDL and DML statements that do not return result sets, such as CREATE TABLE, INSERT, UPDATE, and DELETE. The Query class methods are used to execute SELECT statements.

Exec Class Methods

The following provides a brief example of using the Exec class methods. (For the complete example, refer to **Example 1**.)

Prerequisites: You have connected to a database named **db**.

Step 1 Define a string slice that contains four SQL statements.

```
sqls := []string{
    "DROP TABLE IF EXISTS testExec",
    "CREATE TABLE testExec(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
    "INSERT INTO testExec VALUES(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
    "INSERT INTO testExec VALUES(:f1, :f2, :f3, :f4, :f5)",
}
```

Step 2 Define data slices.

```
intF1 := []int{2, 3, 4, 5, 6}
intF2 := []string{"hello world", "Huawei", "Beijing", "Nanjing", "Research center"}
intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
intF4 := []time.Time{
    time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
    time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
    time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
    time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
    time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
}
intF5 := []bool{false, true, false, true, true}
```

Step 3 Traverse the **sqls** slice using the loop structure and RANGE keyword, and call the Exec method to execute SQL statements that do not return result sets.

```
for _, s := range sqls {
    if strings.Contains(s, ":f") {  // Check whether SQL statements are parameterized.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        for i, _ := range intF1 {  // Traverse data indexes.
            _, err := db.Exec(s, intF1[i], intF2[i], intF3[i], intF4[i], intF5[i])   // Execute the parameterized insert statement.
            if err != nil {
                log.Fatal(err)
            }
        }
    } else {  // Non-parameterized SQL statements.
        _, err = db.Exec(s)  //Execute the DDL statements directly.
        if err != nil {
            log.Fatal(err)
        }
    }
}
```

----End

Query Class Methods

After calling the Exec class methods to insert data into the **testExec** table, you can call the Query class methods to execute SQL statements that return result sets. QueryRow(query string) returns only one data row, whereas Query(query string) returns multiple data rows.

The following provides a brief example of using the Query class methods. (For the complete example, refer to **Example 1**.)

- Call the QueryRow method to query a single row of data.
  ```
  row := db.QueryRow("SELECT * FROM testExec")
  ```
- Call the Query method to execute a query with specified conditions and return multiple rows of data.
  ```
  rows, err := db.Query("SELECT * FROM testExec WHERE f1 > :1", 1)
  if err != nil {
      log.Fatal(err)
  }
  ```

5.7.2.4 Processing the Result Set

After running the SELECT statement to query data, you can use the **type Rows** and **type Row** APIs to process the data result set. The type Row API is used to process single-row results and provides the **(r *Row)Scan(dest...interface{})** method to scan data to a specified variable. The type Rows API is used to process multi-row results and provides the traversal method **(rs *Rows)Next()** and data scanning method **(rs *Rows)Scan(dest...interface{})** to scan data row by row.

Below is a brief example. (For the complete example, refer to example 1 in **Typical Application Development Examples**.)

After calling the **QueryRow** method to query a single row of data, use the Scan method of the type Row API to scan this row of data to the specified variable.

Step 1 Define a variable to receive the data result set.

```
var f1 int
var f2 string
var f3 float64
var f4 time.Time
var f5 bool
```

Step 2 Use the Scan method of the type Row API to scan a single row of data to a specified variable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
err = row.Scan(&f1, &f2, &f3, &f4, &f5)
if err != nil {
   log.Fatal(err)
} else {
   fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
}
```

----End

After calling the **Query** method to return multiple rows of data, use the for loop and the Next and Scan methods of the type Rows API to scan data row by row to the specified variable.

Step 1 Define a variable to receive the data result set.

```
var f1 int
var f2 string
var f3 float64
var f4 time.Time
var f5 bool
```

Step 2 Use the for loop and the Next and Scan methods of the type Rows API to scan data row by row to the specified variable.

```
for rows.Next() {
   err = rows.Scan(&f1, &f2, &f3, &f4, &f5)
   if err != nil {
      log.Fatal(err)
   } else {
      fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
   }
}
```

----End

5.7.2.5 Closing the Database Connection

After completing all necessary operations, you can use the **(db *DB)Close()** method to close the database connection.

Assume that the connection handle is **db**. The procedure for closing the database connection is as follows:

```
db.Close()
```

5.7.3 Typical Application Development Examples

This section demonstrates how to connect to a database, create a table, insert data, and return results. Example 1 uses a single IP address and port number as an example. Example 2 uses multiple IP addresses and port numbers as an example.

Example 1

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
   "database/sql"
   "fmt"
   "os"
   _ [Go driver module name] // Replace it with the actual Go driver module name.
   "log"
   "strings"
   "time"
)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// The code snippet below illustrates the use of a single IP:port pair. This example assumes the IP address, port number, username, and password are stored in environment variables. Before running this example, set the environment variables in your local environment (and adjust the variable names as needed).
func main() {
  hostip := os.Getenv("GOHOSTIP")   // GOHOSTIP indicates the IP address written into the environment variable.
  port := os.Getenv("GOPORT")       // GOPORT indicates the port number written into the environment variable.
  usrname := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the environment variable.
  passwd := os.Getenv("GOPASSWD")   // GOPASSWD indicates the password written into the environment variable.
  str := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + " dbname=gaussdb sslmode=disable" // DSN connection string. You need to set the values of dbname and sslmode.
  //str := "gaussdb://" + usrname + ":" + passwd + "@" + hostip + ":" + port + "/gaussdb?sslmode=disable"  // URL connection string. You need to set the values of the database name and sslmode.
  db, err := sql.Open("gaussdb", str)
  if err != nil {
    log.Fatal(err)  // If an error occurs during execution, use the Fatal method of the standard library log to output the error message, and call os.Exit(1) to terminate the program.
  }
  defer db.Close()  // Delay the execution of db.Close() to ensure that the connection is closed after the database operation is complete.
  err = db.Ping()  // Check whether the database is successfully connected.
  if err != nil {
    log.Fatal(err)
  }

  // Define a string slice that contains four SQL statements.
  sqls := []string{
    "DROP TABLE IF EXISTS testExec",  // Drop the testExec table if it exists.
    "CREATE TABLE testExec(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",  // Create the testExec table.
    "INSERT INTO testExec VALUES(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",  // Insert a data record.
    "INSERT INTO testExec VALUES(:f1, :f2, :f3, :f4, :f5)",  // Parameterize the insert statement.
  }

  // Define data slices.
  intF1 := []int{2, 3, 4, 5, 6}
  intF2 := []string{"hello world", "Huawei", "Beijing", "Nanjing", "Research center"}
  intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
  intF4 := []time.Time{
    time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
    time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
    time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
    time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
    time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
  }
  intF5 := []bool{false, true, false, true, true}

  // Loop structure. Use the RANGE keyword to traverse the sqls slice.
  for _, s := range sqls {
    if strings.Contains(s, ":f") {  // Check whether SQL statements are parameterized.
      for i, _ := range intF1 {  // Traverse data indexes.
        _, err := db.Exec(s, intF1[i], intF2[i], intF3[i], intF4[i], intF5[i])   // Execute the parameterized insert statement.
        if err != nil {
          log.Fatal(err)
        }
      }
    } else {  // Non-parameterized SQL statements.
      _, err = db.Exec(s)  //Execute the DDL statements directly.
      if err != nil {
        log.Fatal(err)
      }
    }
  }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  var f1 int
  var f2 string
  var f3 float64
  var f4 time.Time
  var f5 bool
  row := db.QueryRow("SELECT * FROM testExec")  // Query a single row of data.
  err = row.Scan(&f1, &f2, &f3, &f4, &f5)  // Scan this row of data to the specified variable.
  if err != nil {
    log.Fatal(err)
  } else {
    fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
  }

  rows, err := db.Query("SELECT * FROM testExec WHERE f1 > :1", 1)  // Execute the query with specified
conditions.
  if err != nil {
    log.Fatal(err)
  }
  defer rows.Close()  // Use defer to ensure that rows are closed when the function ends.
  // Use the for loop and the Next and Scan methods of the type Rows API to scan data row by row to the
specified variable.
  for rows.Next() {
    err = rows.Scan(&f1, &f2, &f3, &f4, &f5)
    if err != nil {
      log.Fatal(err)
    } else {
      fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }
  }
}
```

The command output is as follows:

```
f1:1, f2:abcdefg, f3:123.3, f4:2022-02-08 10:30:43.31 +0800 CST, f5:true
f1:2, f2:hello world, f3:641.43, f4:2022-02-08 10:35:43.000623 +0800 CST, f5:false
f1:3, f2:Huawei, f3:431.54, f4:2022-02-10 19:11:54.000353 +0800 CST, f5:true
f1:4, f2:Beijing, f3:5423.52, f4:2022-02-12 06:11:15.000636 +0800 CST, f5:false
f1:5, f2:nanjing, f3:665537.63, f4:2022-02-14 04:51:22.000748 +0800 CST, f5:true
f1:6, f2:Research center, f3:6503.1, f4:2022-02-16 13:45:55.000675 +0800 CST, f5:true
```

Example 2

```
package main
// Set the dependency package based on the dependency package path in the environment.
import (
 "context"
 "database/sql"
 "fmt"
 "os"
  _ [Go driver module name] // Replace it with the actual Go driver module name.
 "log"
 "strings"
 "time"
)
// The code snippet below illustrates the use of multiple IP:port pairs. This example assumes the IP
addresses, port numbers, username, and password are stored in environment variables. Before running this
example, set the environment variables in your local environment (and adjust the variable names as
needed).
func main() {
 ctx := context.Background()
 ctx2SecondTimeout, cancelFunc2SecondTimeout := context.WithTimeout(ctx, 2*time.Second)
 defer cancelFunc2SecondTimeout()
  hostip1 := os.Getenv("GOHOSTIP1")   // GOHOSTIP1 indicates an IP address written into the environment
variable.
  hostip2 := os.Getenv("GOHOSTIP2")   // GOHOSTIP2 indicates an IP address written into the environment
variable.
  hostip3 := os.Getenv("GOHOSTIP3")   // GOHOSTIP3 indicates an IP address written into the environment
variable.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  port1 := os.Getenv("GOPORT1")       // GOPORT1 indicates a port number written into the environment variable.
  port2 := os.Getenv("GOPORT2")       // GOPORT2 indicates a port number written into the environment variable.
  usrname := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the environment variable.
  passwd := os.Getenv("GOPASSWD")   // GOPASSWD indicates the password written into the environment variable.

  str := "host="+hostip1+","+hostip2+","+hostip3+" port="+port1+","+port2+" user="+usrname+" password="+passwd+" dbname=gaussdb sslmode=disable" // DSN connection string. You need to set the values of dbname and sslmode.
  //str := "gaussdb://"+usrname+":"+passwd+"@"+hostip1+":"+port1+","+hostip2+":"+port2+","+hostip3+"/gaussdb?sslmode=disable"  // URL connection string. You need to set the values of the database name and sslmode.
 db, err := sql.Open("gaussdb", str)
 if err != nil {
  log.Fatal(err)
 }
 defer db.Close()  // Delay the execution of db.Close() to ensure that the connection is closed after the database operation is complete.

 // Ping database connection with 2 second timeout
 err = db.PingContext(ctx2SecondTimeout)  // Check whether the database is successfully connected.
 if err != nil {
  log.Fatal(err)
 }

   // Define a string slice that contains four SQL statements.
 sqls := []string{
  "DROP TABLE IF EXISTS testExecContext",
  "CREATE TABLE testExecContext(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",  // Create a table.
  "INSERT INTO testExecContext VALUES(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",  // Insert a data record.
  "INSERT INTO testExecContext VALUES(:f1, :f2, :f3, :f4, :f5)",  // Parameterize the insert statement.
    }

   // Define data slices.
    intF1 := []int{2, 3, 4, 5, 6}
    intF2 := []string{"hello world", "Huawei", "Beijing 2022 Winter Olympics", "nanjing", "Research Center"}
    intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
    intF4 := []time.Time{
        time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
        time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
        time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
        time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
        time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
    }
    intF5 := []bool{false, true, false, true, true}

   // Loop structure. Use the RANGE keyword to traverse the sqls slice.
    for _, s := range sqls {
      if strings.Contains(s, ":f") {  // Check whether SQL statements are parameterized.
            for i, _ := range intF1 {  // Traverse data indexes.
                _, err := db.ExecContext(ctx2SecondTimeout, s, intF1[i], intF2[i], intF3[i], intF4[i], intF5[i])   // Execute the parameterized insert statement.
                if err != nil {
                    log.Fatal(err)
                }
            }
        } else {  // Non-parameterized SQL statements.
            _, err = db.ExecContext(ctx2SecondTimeout, s)  // Execute the DDL statements directly.
            if err != nil {
                log.Fatal(err)
            }
        }
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    var f1 int
    var f2 string
    var f3 float64
    var f4 time.Time
    var f5 bool
    row := db.QueryRowContext(ctx2SecondTimeout, "SELECT * FROM testExecContext")  // Query a single row of data.
        err = row.Scan(&f1, &f2, &f3, &f4, &f5)  // Scan this row of data to the specified variable.
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }

    rows, err := db.QueryContext(ctx2SecondTimeout, "SELECT * FROM testExecContext WHERE f1 > :1", 1)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()  // Use defer to ensure that rows are closed when the function ends.

      // Use the for loop and the Next and Scan methods of the type Rows API to scan data row by row to the specified variable.
    for rows.Next() {
        err = rows.Scan(&f1, &f2, &f3, &f4, &f5)
        if err != nil {
            log.Fatal(err)
        } else {
            fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
        }
    }
}
```

The command output is as follows:

```
f1:1, f2:abcdefg, f3:123.3, f4:2022-02-08 10:30:43.31 +0800 CST, f5:true
f1:2, f2:hello world, f3:641.43, f4:2022-02-08 10:35:43.000623 +0800 CST, f5:false
f1:3, f2:Huawei, f3:431.54, f4:2022-02-10 19:11:54.000353 +0800 CST, f5:true
f1:4, f2:Beijing 2022 Winter Olympics, f3:5423.52, f4:2022-02-12 06:11:15.000636 +0800 CST, f5:false
f1:5, f2:nanjing, f3:665537.63, f4:2022-02-14 04:51:22.000748 +0800 CST, f5:true
f1:6, f2:Research center, f3:6503.1, f4:2022-02-16 13:45:55.000675 +0800 CST, f5:true
```

5.7.4 Go APIs

5.7.4.1 type DB

The type DB API is one of the core APIs in the **database/sql** package. It defines a series of methods for executing SQL queries and managing transactions, as described in **Table 5-119**.

Table 5-119 Common methods of the type DB API

Method	Description	Return Type
(db *DB)Begin()	Starts a transaction. The isolation level of the transaction is determined by the driver.	*Tx and error

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description	Return Type
(db *DB)BeginTx(ctx context.Context, opts *TxOptions)	Starts a transaction with a specified transaction isolation level. A specified context is used until the transaction is committed or rolled back. Once the context is canceled, the **database/sql** package will roll back the transaction.	*Tx and error
(db *DB)Close()	Closes the database and releases all the opened resources.	error
(db *DB)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(db *DB)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(db *DB)Ping()	Checks whether the database connection is still valid and establishes a connection if necessary.	error
(db *DB)PingContext(ctx context.Context)	Checks whether the database connection is still valid in a specified context and establishes a connection if necessary.	error
(db *DB)Prepare(query string)	Creates a prepared statement for subsequent queries or executions.	*Stmt and error
(db *DB)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions in a specified context.	*Stmt and error
(db *DB)Query(query string, args ...interface{})	Executes a query and returns multiple rows of data.	*Rows and error
(db *DB)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query and returns multiple rows of data in a specified context.	*Rows and error
(db *DB)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(db *DB)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

Table 5-120 Parameters of common methods

Parameter	Description
ctx	Specified context.
query	Executed SQL statement.
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples**.
opts	Transaction isolation level and transaction access mode. You can set the transaction isolation level (**opts.Isolation**) to **sql.LevelReadUncommitted**, **sql.LevelReadCommitted**, **sql.LevelRepeatableRead**, or **sql.LevelSerializable**. The transaction access mode (opts.ReadOnly) can be true (read only) or false (read write).

NOTICE

- The Query class methods—Query(), QueryContext(), QueryRow(), and QueryRowContext()—are typically used in query statements, such as SELECT. To prevent unexpected results, refrain from using the Query class methods for non-query statements, such as UPDATE and INSERT. Instead, utilize the Exec() class methods for operation statements.
- After executing a query statement using a Query class method, be sure to retrieve the result using the Next() method provided in **type Rows**. Failure to do so may result in unexpected errors.

Examples

```
// Prerequisites: You have connected to the database using the connection object db.

 // Create a table.
 _, err = db.Exec("CREATE TABLE test_bound(id int, name text)")

 // Binding by location
 _, err = db.Exec("INSERT INTO test_bound(id, name) VALUES(:1, :2)", 1, "Zhang San")
 if err != nil {
  log.Fatal(err)
 }

 // Binding by name
 _, err = db.Exec("INSERT INTO test_bound(id, name) VALUES(:id, :name)", sql.Named("id", 1),
sql.Named("name", "Zhang San"))
 if err != nil {
  log.Fatal(err)
 }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.7.4.2 type Stmt

The type Stmt API is used to prepare SQL statements. You can find its common methods in **Table 5-121**.

Table 5-121 Common methods of the type Stmt API

Method	Description	Return Type
(s *Stmt)Close()	Closes a specified prepared statement.	error
(s *Stmt)Exec(args ...interface{})	Executes a prepared statement with specified parameters and returns a **Result** value. The Prepare-Bind-Execute (PBE) feature is supported. PBE is a method of sending and executing queries. A DN can receive PBE packets through complex query protocols to execute statements. **NOTE** • The prepared statements can use the dollar sign ($) or question mark (?) as placeholders. • The number of placeholders in a prepared statement is determined by the database. When the number of table columns exceeds the database limit or does not match the number of current table columns, the server will return an error. • The PBE feature supports creation, deletion, and modification, allowing U packets to reach a maximum length of 1,023 MB during batch operations. Exceeding this length limit will result in the following error: `bind message length XXX too long. This can be caused by very large or incorrect length specifications on InputStream parameters.` • When inserting a single record, you are advised to use the single-insert API (conn.simpleExec) rather than PBE. This is because the performance of PBE significantly decreases compared to the single-insert API in this scenario. • Reconstructing the driver's error processing results in a performance decrease of less than 5% for PBE.	Result and error
(s *Stmt)ExecContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns a **Result** value.	Result and error

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

(s *Stmt)Query(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryRow(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Row** as the query result.	*Row
(s *Stmt)QueryRowContext (ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Row** as the query result.	*Row

Parameters

Table 5-122 Parameters of common methods

Parameter	Description
ctx	Specified context.
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples**.

NOTICE

1. The Query class methods—Query(), QueryContext(), QueryRow(), and QueryRowContext()—are typically used in query statements, such as SELECT. The Exec() class methods are used for executing operation statements. If the Query class methods are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the Query class methods to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement by using a Query class method needs to be obtained through the Next() method in **type Rows**. If the result is not obtained through the Next() method, unexpected errors may occur.

5.7.4.3 type Tx

The type Tx API defines a series of methods for managing database transactions. You can find its common methods in **Table 5-123**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-123 Common methods of the type Tx API

Method	Description	Return Type
(tx *Tx)Commit()	Commits a transaction.	error
(tx *Tx)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(tx *Tx)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(tx *Tx)Prepare(query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back.	*Stmt and error
(tx *Tx)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back. The specified context will be used in the preparation phase, not in the transaction execution phase. The statement returned by this method will be executed in the transaction context.	*Stmt and error
(tx *Tx)Query(query string, args ...interface{})	Executes a query that returns rows of data.	*Rows and error
(tx *Tx)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns rows of data in a specified context.	*Rows and error

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

(tx *Tx)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(tx *Tx)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row
(tx *Tx)Rollback()	Rolls back a transaction.	error

(tx *Tx)Stmt(stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement.
	Example:
	`str, err := db.Prepare("INSERT INTO t1 VALUES(:1, :2)")`
	`tx, err := db.Begin()`
	`res, err := tx.Stmt(str).Exec(1, "aaa")`
		*Stmt

(tx *Tx)StmtContext(ctx context.Context, stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement in a specified context.	*Stmt

Parameters

Table 5-124 Parameters of common methods

Parameter	Description
ctx	Specified context.
query	Executed SQL statement.
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples**.
stmt	Existing prepared statement, which is generally the prepared statement returned by the prepared statement

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- The Query class methods—Query(), QueryContext(), QueryRow(), and QueryRowContext()—are typically used in query statements, such as SELECT. To prevent unexpected results, refrain from using the Query class methods for non-query statements, such as UPDATE and INSERT. Instead, utilize the Exec() class methods for operation statements.
- After executing a query statement using a Query class method, be sure to retrieve the result using the Next() method provided in **type Rows**. Failure to do so may result in unexpected errors.

5.7.4.4 type Rows

The type Rows API is used to process multi-row results. You can find its methods such as traversal and data scanning in **Table 5-125**.

Table 5-125 Common methods of the type Rows API

Method	Description	Return Type
(rs *Rows)Close()	Closes **Rows** to stop the iteration of the data set.	error
(rs *Rows)ColumnTypes()	Returns the column type.	[]*ColumnType and error
(rs *Rows)Columns()	Returns the column name.	[]string and error
(rs *Rows)Err()	Returns any errors that occur during iteration.	error
(rs *Rows)Next()	Prepares the next data row to be read with the Scan method. If the result set of the next data row exists, **true** is returned. Otherwise, **false** is returned.	Boolean
(rs *Rows)Scan(dest ...interface{})	Copies columns of the currently traversed rows of data to the value specified by **dest**.	error
(rs *Rows)NextResultSet()	Specifies whether there is an additional result set.	Boolean

Parameters

Table 5-126 Parameters of common methods

Parameter	Description

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

dest	The column to be queried needs to be copied to the value specified by this parameter.

5.7.4.5 type Row

The type Row API is used to process single-row results. You can find its methods such as data scanning in **Table 5-127**.

Table 5-127 Common methods of the type Row API

Method	Description	Return Type
(r *Row)Scan(dest ...interface{})	Copies the columns in the current row of data to the value specified by **dest**.	error
(r *Row)Err()	Returns errors that occur during execution.	error

Parameters

Table 5-128 Parameters of common methods

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

5.7.4.6 type ColumnType

The type ColumnType API is used to describe the column types in a database table. You can find its common methods in **Table 5-129**.

Table 5-129 Common methods of the type ColumnType API

Method	Description	Return Type
(ci *ColumnType)DatabaseTypeName()	Returns the name of the column-type database system. If an empty string is returned, driver-type names are not supported.	error

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

(ci *ColumnType)DecimalSize()	Returns the scale and precision of the decimal type. If the value of **ok** is **false**, the specified type is unavailable or not supported.	precision, scale int64, ok boolean
(ci *ColumnType)Length()	Returns the length of the data column type. If the return value of **ok** is **false**, the specified type is of non-variable length.	length int64, ok boolean
(ci *ColumnType)ScanType()	Returns a Go type that can be used for scanning by using Rows.Scan.	reflect.Type
(ci *ColumnType)Name()	Returns the name of a data column.	string

5.7.4.7 type Result

The type Result API is used to obtain the result of executing an SQL statement. You can find its common methods in **Table 5-130**.

Table 5-130 Common methods of the type Result API

Method	Description	Return Type
(res Result)RowsAffected()	Obtains the number of affected rows. It is useful when you perform modification operations such as UPDATE and DELETE. It helps you know the number of rows that are successfully modified.	int64 and error

5.8 Development Based on ECPG

5.8.1 Development Process

Figure 5-20 shows the general development process based on ECPG.

This process involves these steps: prepare components for ECPG-based development, develop embedded SQL-C programs, preprocess embedded SQL-C programs, compile C programs, and execute C programs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-20 General development process based on ECPG

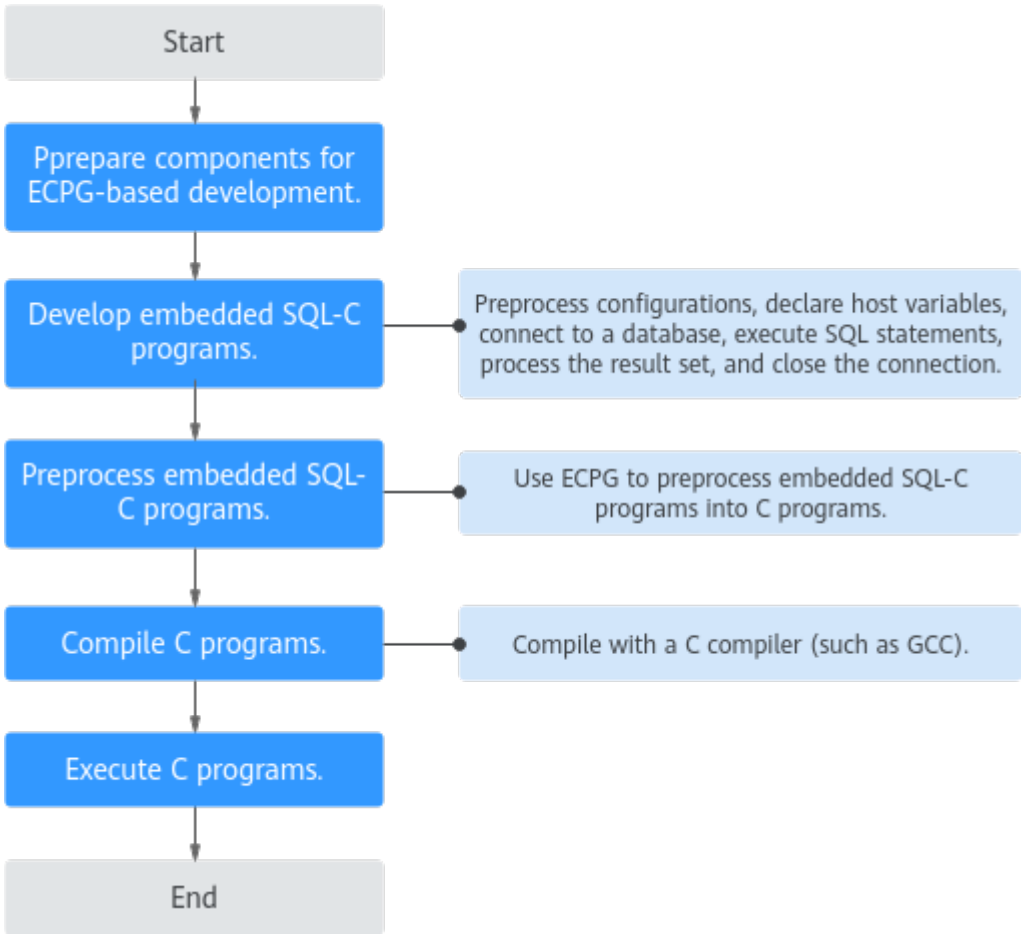
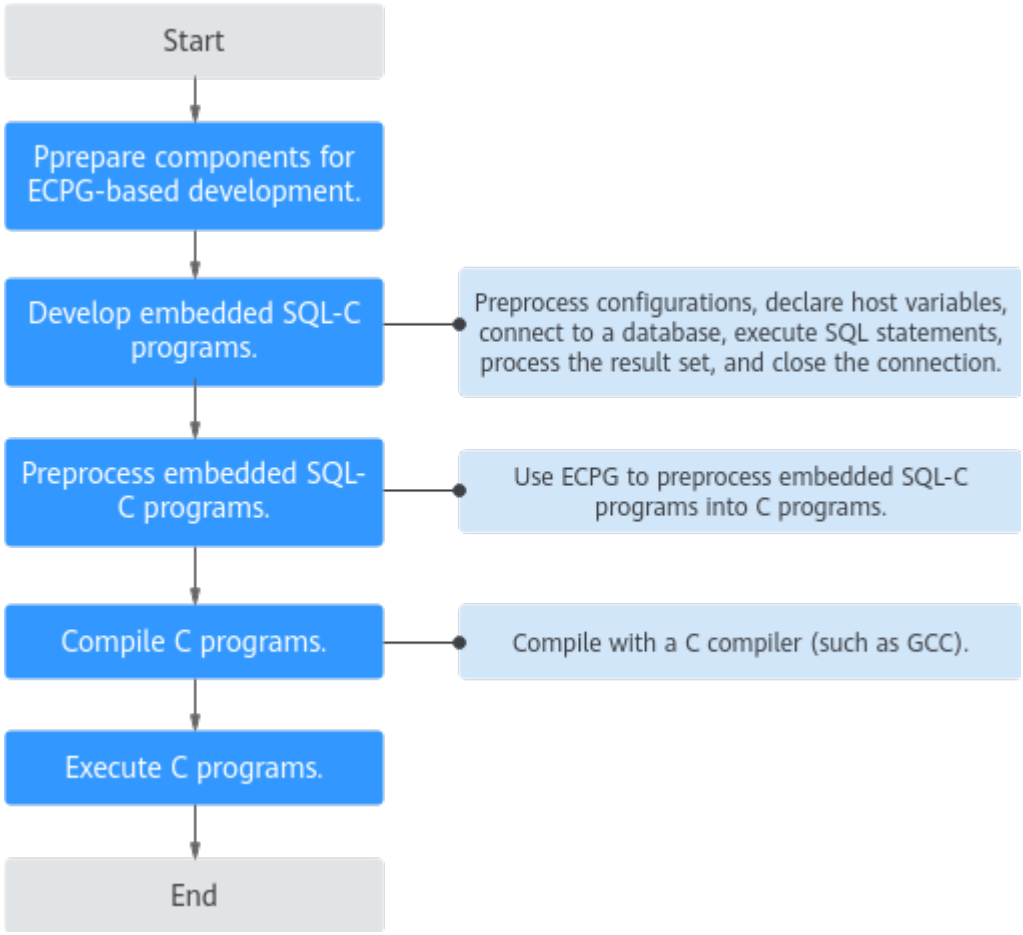


Figure 5-21 shows the process of developing an embedded SQL-C program.

The process consists of preprocessing configuration, declaring host variables, managing database connections, executing SQL statements, processing result sets, and closing connections.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-21 Embedded SQL-C program development process



5.8.2 Preparing Components for ECPG-based Development

Before developing applications based on ECPG, ensure that the GaussDB client components and compilation toolchain have been correctly configured in your development environment.

ECPG-based development depends on specific preprocessors, dynamic link libraries, and header files. The following table outlines the paths and compilation reference rules for core components.

Table 5-131 Paths and compilation reference rules for core ECPG components

Component Category	Core File/Library Name	Path	Compilation/ Reference Parameter	Description
Preprocessor	ecpg	$GAUSSHOME/bin	None	Converts .pgc source code files into standard .c files. It is a binary tool.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Component Category	Core File/Library Name	Path	Compilation/ Reference Parameter	Description
Communication and control library	**libecpg.so**, **libecpg.so.6**, and **libecpg.so.6.4**	$GAUSSHOME/lib	-lecpg	Provides underlying implementation for database connection management, SQL statement execution, and transaction control.
Type processing library	**libpgtypes.so**, **libpgtypes.so.6**, and **libpgtypes.so.6.4**	$GAUSSHOME/lib	-lpgtypes	Supports calculation and conversion for complex data types such as numeric, date, timestamp, and interval.
Dependency header file	**sqlca.h**, **sqlda.h**, and more	$GAUSSHOME/include/ecpg	-I $GAUSSHOME/include/ecpg	Provides the macro definitions and data structure declarations required during precompilation.

Tips for preparing compilation preprocessing tools:

- ECPG-based development requires the GNU Compiler Collection (GCC). Version 7.3.0 is recommended.
- The ECPG client tool supports the same OSs and CPU architectures (x86/Arm) as specified in the server's published support matrix.

Known issue and solution:

Dynamic library addressing during runtime: By default, OSs do not scan the GaussDB library directory. Before running compiled binary files, you must add *$GAUSSHOME*/**lib** to the system environment variable *LD_LIBRARY_PATH*.

5.8.3 Embedded SQL-C Program Development

5.8.3.1 Foundational Concepts

Embedded SQL-C application development is a programming approach that uses the ECPG precompiler to embed SQL statements directly into standard C code. This approach enables developers to efficiently build database applications by combining C's procedural control capabilities with SQL's data processing capabilities within a single source file.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Before writing embedded code, understanding the following foundational concepts will help you better structure your applications.

C Preprocessor and ECPG Precompiler

In embedded development, your source code (.pgc) undergoes two serial and independent preprocessing stages. Understanding their distinct roles is essential:

1. The ECPG precompiler runs first. It scans the source code for statements prefixed with **EXEC SQL**, converts them into standard C functions that call underlying libraries, and generates a temporary .c file.
2. The standard C preprocessor (part of the C compiler) then takes over the generated .c file, processing native C directives such as **#include** for header files and **#define** for macros.

For detailed functionalities of the ECPG precompiler, refer to **ECPG-based Preprocessing, Compilation, and Execution**.

Static SQL and Dynamic SQL

Most applications are designed to work with static SQL statements. In such cases, the specific SQL commands to be executed, along with the tables and columns involved, are predetermined before compilation. However, certain applications—such as general-purpose query tools—may not determine the exact SQL structure until runtime. Dynamic SQL is an advanced technique that allows applications to dynamically construct, accept, and execute SQL statements at runtime.

Host Variables and Data Types

Host variables are variables declared in C applications to transfer actual data between applications and databases. For example, database query results can be directly fetched into host variables, or values stored in host variables can be inserted as parameters into database tables. Host variables serve as a bridge linking the C language data space with SQL statement execution.

Due to discrepancies between native data types in the C language and SQL data types in databases, the use of host variables relies heavily on type mapping mechanisms. Developers must not only employ host variables to fetch or insert data but also adhere to strict mapping rules to safely convert special data types in databases (such as timestamp and string) into primitive or non-primitive data types in C (such as structure and array).

Cursors

A database query (SELECT) typically returns multiple rows of data, whereas a regular C variable can hold only a single value at any given time. Cursors are the mechanism designed to resolve this discrepancy. By declaring a cursor, your application opens a "data window" over the result set and then sequentially fetches and processes data row by row.

Descriptor Area (SQLDA)

When you employ the most sophisticated dynamic SQL techniques—where not only the exact SQL statement is unknown at coding time, but even the number

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

and data types of returned columns are uncertain—ordinary host variables are insufficient. This is where the descriptor area (SQLDA) comes into play. It is an advanced in-memory data structure that enables applications to dynamically parse, allocate, and store data of unknown structures at runtime.

Error Handling Mechanisms

After execution, an embedded SQL statement may succeed, fail, or generate a warning. ECPG provides two primary mechanisms to handle these outcomes: One is the SQL communication area (SQLCA), which is a global data structure that conveys runtime status feedback (such as error codes and the number of affected rows) to applications. Based on this feedback, the applications can perform different operations accordingly. The other mechanism is the WHENEVER statement, which allows you to specify automatic actions—such as jumping to a specific code segment, calling a function, or exiting the application directly—when the system detects an error or warning.

5.8.3.2 Declaring Host Variables

A host variable is a bridge between a C program and SQL and must be declared in a specific code block.

The syntax is as follows:

The declaration section starts with the following code:

```
EXEC SQL BEGIN DECLARE SECTION;
```

And ends with:

```
EXEC SQL END DECLARE SECTION;
```

Between them, there must be regular C variable declarations. An example is as follows:

```
int   x = 4;
char  foo[16], bar[16];
```

NOTICE

- The type of the host variables declared between the start and end of the marked section must be one of the supported data types. For details, see **Table 5-145**.
- For simple single variables, you can omit the full DECLARE section by using a directive prefix for implicit declaration, for example, **EXEC SQL int i = 4**.
- Variables not used within SQL commands may be declared outside the DECLARE section. Only variables declared inside the DECLARE section are usable in SQL statements.
- If a host variable involves a user-defined structure or union, its complete type definition must also be listed inside the DECLARE section. Otherwise, ECPG cannot resolve its underlying memory structure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.3 Managing Database Connections

Connecting to a Database

You can run the following command to connect to the database:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

The target can be declared using the following methods. The italic part is a variable. Change it based on the actual situation.

- *dbname*[@*hostname*][:*port*]
- tcp:gaussdb://*hostname*[:*port*][/*dbname*][?*options*]
- unix:gaussdb://*hostname*[:*port*][/*dbname*][?*options*]

ECPG supports IPv6 formats but requires *hostname* to be enclosed in square brackets ([]).

Example IPv6 formats:

```
db1@[::1]
db1@[::1]:5432
tcp:gaussdb://[::1]/db1
tcp:gaussdb://[::1]:5432/db1
unix:gaussdb://[::1]/db1
unix:gaussdb://[::1]:5432/db1
```

NOTE

Currently, ECPG supports the following IPv6 formats:

- Uppercase abbreviated format
- Lowercase abbreviated format
- Uppercase full format
- Lowercase full format

The IPv6 loopback address is **::1** or **0:0:0:0:0:0:0:1**. Therefore, UDS supports only these two IPv6 formats.

Methods of declaring a connection username are as follows:

- *username*/*password*
- *username* SQLIDENTIFIED BY *password*
- *username* USING *password*

As mentioned above, the **username** and **password** parameters can be an SQL identifier, an SQL string, or a reference to a character variable.

connection-name indicates a connection name. If a program uses only one connection, **connection-name** can be omitted. If a program uses multiple connections, the latest connection is used as the current connection.

NOTE

- If **ip-port** is specified in a connection statement, username and password must be specified. This rule is determined by the kernel communication authentication. If **ip-port** is not specified, the local *$PGPORT* (UDS) is used for communication.
- If the SSL protocol is used for client connection, run the **tcp:gaussdb://***hostname***[:***port***][/***dbname***][?***options***]** command and set **sslmode** to **disable** or **require** in *options*.

An example is as follows:

```
#include <stdlib.h>
EXEC SQL CONNECT TO mydb@sql.mydomain.com;

EXEC SQL CONNECT TO unix:gaussdb://sql.mydomain.com/mydb AS myconnection USER username;

EXEC SQL BEGIN DECLARE SECTION;

/* The values of target, user, and passwd must be read from environment variables or configuration files.
Environment variables need to be configured as required. If no environment variable is used, a character
string can be directly assigned. */
const char *target = getenv("EXAMPLE_TARGET_ENV");
const char *user = getenv("EXAMPLE_USERNAME_ENV");
const char *passwd = getenv("EXAMPLE_PASSWD_ENV");
EXEC SQL END DECLARE SECTION;
...

EXEC SQL CONNECT TO :target USER :user USING :passwd;
/* or EXEC SQL CONNECT TO :target USER :user/:passwd; */
```

For details about the complete usage example, see the example of using the connection syntax in **CONNECT**.

NOTE

- The format of the connection target is not described in the SQL standard. Therefore, to develop a portable application, you can use the method in the last example to encapsulate the connection target string into a variable.
- For details about the ECPG compatibility, see **ECPG Compatibility**.

Managing Connections

By default, statements in embedded SQL-C applications execute on the current connection—that is, the most recently established connection. When an application needs to handle multiple database instances concurrently, connection routing can be implemented through these two methods:

- Method 1: Explicitly specify a connection for each SQL statement.
  ```
  EXEC SQL AT connection-name SELECT ...;
  ```
  This method is particularly suitable if the application needs to use several connections in a mixed order.
- Method 2: Execute a statement to switch the connection.
  ```
  EXEC SQL SET CONNECTION connection-name;
  ```
  This method is particularly suitable if many statements are executed on the same connection.

The following is an example of managing connections.

```
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
   char dbname[1024];
EXEC SQL END DECLARE SECTION;

int main()
{
    EXEC SQL CONNECT TO testdb1 AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb2 AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb3 AS con3 USER testuser;

    /* This query will be executed in the most recently connected database testdb3. */
    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb3)\n", dbname);

    /* Use AT to run a query in testdb2. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL AT con2 SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb2)\n", dbname);

    /* Switch to connection to testdb1. */
    EXEC SQL SET CONNECTION con1;

    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb1)\n", dbname);

    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

The result is as follows:

```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

NOTE

- In multi-threaded mode, different threads must not share the same connection name—each thread requires a unique connection name.
- Connections must be established and closed within the same process or thread.
- If an application creates multiple execution threads, they cannot share the same connection. Access to connections must be explicitly controlled (for example, using mutexes), or each thread should maintain its own dedicated connection.

5.8.3.4 Executing SQL Statements

Embedded SQL-C programs support direct execution of static SQL statements (such as SELECT and INSERT) and dynamic SQL statements. For details about dynamic SQL statements, see **Executing Dynamic SQL Statements**. This section describes how to execute static SQL statements.

To execute a static SQL statement, follow these steps:

Step 1 Create a table definition.

```
EXEC SQL CREATE TABLE foo (a int, b varchar);
```

Step 2 Insert a single record.

```
EXEC SQL INSERT INTO foo VALUES (5, 'abc');
```

Step 3 Delete a specified record.

```
EXEC SQL DELETE FROM foo WHERE a = 5;
```

Step 4 Update table data.

```
EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
```

Step 5 Query data in a single row.

```
EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
```

The INTO keyword directly maps the query result to the bound host variable.

----End

A complete example is as follows:

Prerequisites: The **testdb** database has been created.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
int main ()
{
    ECPGdebug (1, stderr);

    EXEC SQL BEGIN DECLARE SECTION;
    int var_a;
    EXEC SQL END DECLARE SECTION;

    /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb;

    /* Create a table. */
    EXEC SQL CREATE TABLE foo (a int, b varchar);

    /* Insert data. */
    EXEC SQL INSERT INTO foo VALUES (5, 'abc');
    EXEC SQL INSERT INTO foo VALUES (6, 'def');
    EXEC SQL INSERT INTO foo VALUES (7, 'ghi');

    /* Delete a row. */
    EXEC SQL DELETE FROM foo WHERE a = 5;

    /* Update table data. */
    EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;

    /* Query table data in a single row. */
    EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';

    /* Print the query results. */
    printf("select res is %d\n", var_a);
    EXEC SQL DISCONNECT;
    return 0;
}
```

The command output is as follows (**[39841]** is a randomly generated thread ID):

```
[39841]: ECPGdebug: set to 1
[39841]: ECPGconnect: opening database testdb on <DEFAULT> port <DEFAULT>
[39841]: ecpg_execute on line 17: query: create table foo ( a int , b varchar ); with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 17: using PQexec
[39841]: ecpg_execute on line 17: OK: CREATE TABLE
[39841]: ecpg_execute on line 20: query: insert into foo values ( 5 , 'abc' ); with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 20: using PQexec
[39841]: ecpg_execute on line 20: OK: INSERT 0 1
[39841]: ecpg_execute on line 21: query: insert into foo values ( 6 , 'def' ); with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 21: using PQexec
[39841]: ecpg_execute on line 21: OK: INSERT 0 1
[39841]: ecpg_execute on line 22: query: insert into foo values ( 7 , 'ghi' ); with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 22: using PQexec
[39841]: ecpg_execute on line 22: OK: INSERT 0 1
[39841]: ecpg_execute on line 25: query: delete from foo where a = 5; with 0 parameter(s) on connection
testdb
[39841]: ecpg_execute on line 25: using PQexec
[39841]: ecpg_execute on line 25: OK: DELETE 1
[39841]: ecpg_execute on line 28: query: update foo set b = 'gdp' where a = 7; with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 28: using PQexec
[39841]: ecpg_execute on line 28: OK: UPDATE 1
[39841]: ecpg_execute on line 31: query: select a from foo where b = 'def'; with 0 parameter(s) on
connection testdb
[39841]: ecpg_execute on line 31: using PQexec
[39841]: ecpg_execute on line 31: correctly got 1 tuples with 1 fields
[39841]: ecpg_get_data on line 31: RESULT: 6 offset: 4; array: no
select res is 6
[39841]: ecpg_finish: connection testdb closed
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Transaction Management

Transaction management is generally involved in SQL statement execution. A transaction is the basic unit of database operations, which ensures that a group of SQL statements are either all successfully executed or all rolled back upon failure. The ACID properties of a transaction are as follows:

- Atomicity: All operations in a transaction are either completed or not completed.
- Consistency: The database status is consistent before and after a transaction is executed.
- Isolation: Multiple transactions are executed concurrently without interfering with each other.
- Durability: After a transaction is committed, modifications to the database are permanent.

In ECPG default mode, a statement is committed only when **EXEC SQL COMMIT** is issued. Embedded SQL APIs support automatic transaction commit (which is enabled by the EXEC SQL SET AUTOCOMMIT TO ON statement). In autocommit mode, each command is automatically committed unless it is inside an explicit transaction block. You can disable autocommit mode using the EXEC SQL SET AUTOCOMMIT TO OFF statement.

Common transaction management commands are as follows:

- **EXEC SQL COMMIT**: commits an ongoing transaction.
- **EXEC SQL ROLLBACK**: rolls back an ongoing transaction.
- **EXEC SQL SET AUTOCOMMIT TO ON**: enables autocommit mode.
- **EXEC SQL SET AUTOCOMMIT TO OFF**: disables autocommit mode. This is the default mode.

5.8.3.5 Processing the Result Set

- SELECT statements that return single-row results can be directly executed using EXEC SQL. For details, see **Executing SQL Statements**.

  Example:

  ```
  /* Create a table and insert data. */
  EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);

  /* The query result is in a single row. :num is the host variable. */
  EXEC SQL SELECT number1 INTO :num FROM test_table WHERE number2 = 1;
  ```

- To process a multi-row result set, you must use a cursor. For details about how to use a cursor, see **Using Cursors**.

  Example:

  ```
  /* Create a table and insert data. */
  EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (3, 1);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (4, 1);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (5, 1);

  /* Define host variables. */
  EXEC SQL BEGIN DECLARE SECTION;
  int v1;
  int v2;
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
EXEC SQL END DECLARE SECTION;

    /* Declare a cursor. */
EXEC SQL DECLARE test_bar CURSOR FOR SELECT number1, number2 FROM test_table ORDER BY
number1;

    /* Open the cursor. */
EXEC SQL OPEN test_bar;

    /* When the cursor reaches the end of the result set, exit the loop. */
EXEC SQL WHENEVER NOT FOUND DO BREAK;

    /* Obtain the query result set. */
while(1)
{
    EXEC SQL FETCH NEXT FROM test_bar INTO :v1, :v2;
    printf("number1 = %d, number2 = %d\n",v1,v2);
}

    /* Close the cursor. */
EXEC SQL CLOSE test_bar;
```

Using Cursors

To retrieve a multi-row result set, an application has to declare a cursor and fetch each row from the cursor.

Step 1 Declare a cursor.

```
EXEC SQL DECLARE c CURSOR FOR SELECT * FROM tb1;
```

Step 2 Open the cursor.

```
EXEC SQL OPEN c;
```

Step 3 Fetch a row of data from the cursor.

```
EXEC SQL FETCH 1 IN c INTO :a, :str;
```

Step 4 Close the cursor.

```
EXEC SQL CLOSE c;
```

----End

NOTE

When the GUC parameter **enable_ecpg_cursor_duplicate_operation** is enabled (it is enabled by default) and ECPG is used to connect to an ORA-compatible database, you can repeatedly open and close the cursor in the following ways:

```
EXEC SQL OPEN c;
EXEC SQL OPEN c;
EXEC SQL CLOSE c;
EXEC SQL CLOSE c;
```

For details about how to use the cursor, see **DECLARE**. For details about the **FETCH** command, see "SQL Reference > SQL Syntax > F > FETCH" in *Reference*.

A complete example is as follows:

Prerequisites: The **testdb** database has been created.

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int *a = NULL;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    char *str = NULL;
    EXEC SQL END DECLARE SECTION;

    int count = 0;

    /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb ;
    EXEC SQL SET AUTOCOMMIT TO OFF;
    EXEC SQL BEGIN;
    EXEC SQL DROP TABLE IF EXISTS tb1;
    EXEC SQL CREATE TABLE tb1(id int, info text);
    EXEC SQL INSERT INTO tb1 (id, info) SELECT generate_series(1, 100000), 'test';
    EXEC SQL SELECT COUNT(*) INTO :a FROM tb1;
    printf ("a is %d\n", *a);
    EXEC SQL COMMIT;

    /* Define a cursor. */
    EXEC SQL DECLARE c CURSOR FOR SELECT * FROM tb1;

    /* Open the cursor. */
    EXEC SQL OPEN c;
    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while(1) {
        /* Fetch data. */
        EXEC SQL FETCH 1 IN c INTO :a, :str;
        count++;
        if (count == 100000) {
            printf("Fetch res: a is %d, str is %s", *a, str);
        }
    }

    /* Close the cursor. */
    EXEC SQL CLOSE c;
    EXEC SQL SET AUTOCOMMIT TO ON;
    EXEC SQL DROP TABLE tb1;
    EXEC SQL DISCONNECT;

    ECPGfree_auto_mem();
    return 0;
}
```

The result is as follows:

```
a is 100000
Fetch res: a is 100000, str is test
```

When the cursor points to a row in a table, you can use the WHERE CURRENT OF cursor_name syntax to update or delete the row. For detailed restrictions, refer to the description of this syntax in "SQL Reference > SQL Syntax > U > UPDATE" in *Reference*.

A complete example is as follows:

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int va;
    int vb;
    EXEC SQL END DECLARE SECTION;

    int count = 0;

    /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb;
    EXEC SQL SET AUTOCOMMIT TO OFF;
    EXEC SQL BEGIN;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL DROP TABLE IF EXISTS t1;
    EXEC SQL CREATE TABLE t1(c1 INT, c2 INT);
    EXEC SQL INSERT INTO t1 VALUES(generate_series(1, 10000), generate_series(1, 10000));
    EXEC SQL COMMIT;

    /* Define a cursor. */
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT * FROM t1 WHERE c1 < 100 FOR UPDATE;

    /* Open the cursor. */
    EXEC SQL OPEN cur1;

    /* Fetch data. */
    EXEC SQL FETCH 1 IN cur1 INTO :va, :vb;
    printf("c1: %d, c2: %d\n", va, vb);

    /* Use WHERE CURRENT OF to delete the current row. */
    EXEC SQL DELETE FROM t1 WHERE CURRENT OF cur1;

    /* Continue to fetch data. */
    EXEC SQL FETCH 1 IN cur1 INTO :va, :vb;
    EXEC SQL FETCH 1 IN cur1 INTO :va, :vb;
    printf("c1: %d, c2: %d\n", va, vb);

    /* Use WHERE CURRENT OF to update the current row. */
    EXEC SQL UPDATE t1 SET c2 = 21 WHERE CURRENT OF cur1;

    /* Query the updated data. */
    EXEC SQL SELECT c2 INTO :vb FROM t1 WHERE c1 = :va;
    printf("c1: %d, c2: %d\n", va, vb);

    /* Close the cursor. */
    EXEC SQL CLOSE cur1;

    /* Restore automatic commit and clear data. */
    EXEC SQL SET AUTOCOMMIT TO ON;
    EXEC SQL DROP TABLE t1;
    EXEC SQL DISCONNECT;

    ECPGfree_auto_mem();
    return 0;
}
```

The result is as follows:

```
c1: 1, c2: 1
c1: 3, c2: 3
c1: 3, c2: 21
```

5.8.3.6 Closing a Connection

Upon completion of service logic, database connections must be explicitly closed to release underlying resources and prevent server connection leaks.

To close a connection, use the following statement:

```
EXEC SQL DISCONNECT [connection];
```

Based on different concurrency or multi-instance management requirements, the **[connection]** parameter can be declared in the following ways:

- **connection-name**: closes the named database connection precisely.
- **default**: closes the unnamed connection established by default.
- **current**: closes the most recently active connection in the application.
- **all**: safely closes all GaussDB session connections (established by the current application) at once.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.7 Embedded SQL Syntax Reference

5.8.3.7.1 ALLOCATE DESCRIPTOR

Description

Allocates a newly named SQL descriptor area.

Syntax

```
ALLOCATE DESCRIPTOR name
```

Parameters

name

SQL descriptor name. It is case-sensitive and is an SQL identifier or a host variable.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

DEALLOCATE DESCRIPTOR, **GET DESCRIPTOR**, and **SET DESCRIPTOR**

5.8.3.7.2 CONNECT

Description

Establishes a connection between the client and the SQL server.

Syntax

```
CONNECT TO connection_target [ AS connection_name ] [ USER connection_user ]
```

Parameters

- **connection_target**

  Specifies the target server to be connected in one of the following formats:
  - [*database_name*] [@ *host*] [: *port*]: connection over TCP/IP.
  - unix:gaussdb://*host* [: *port*] / [*database_name*] [? *connection_option*]: connection over UDSs.
  - tcp:gaussdb://*host* [:*port*] / [*database_name*] [?*connection_option*]: connection over TCP/IP.
  - SQL string constant: a value in any of the preceding formats.

  NOTE

  For details about **connection_target**, see **ECPG Compatibility**.

- **connection_name**

  An optional identifier used for the connection, which can be referenced in other commands. It can be an SQL identifier or a host variable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **connection_user**

  Username and password for connecting to a database.

  You can select one of the following methods:

  a. *user_name*/*password*

  b. *user_name* SQLIDENTIFIED BY *password*

  c. *user_name* USING *password*

  Alternatively, you can use this parameter to specify the username and password.

  The username and password can be SQL identifiers, string constants, or host variables.

NOTE

In the preceding parameters, the information in italics refers to variables. Replace them based on the actual situation.

Examples

Here are several variants of specifying connection parameters:

```
EXEC SQL CONNECT TO "connectdb" AS main;
EXEC SQL CONNECT TO "connectdb" AS second;
EXEC SQL CONNECT TO 'connectdb' AS main;
EXEC SQL CONNECT TO REGRESSDB1 as main;
EXEC SQL CONNECT TO connectdb AS :id;
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb;
EXEC SQL CONNECT TO connectdb AS main USER connectuser USING "connectdb";
EXEC SQL CONNECT TO connectdb AS main;
EXEC SQL CONNECT TO tcp:gaussdb://localhost/connectdb USER connectuser IDENTIFIED BY connectpw;
EXEC SQL CONNECT TO tcp:gaussdb://localhost:$PORT/connectdb USER connectuser SQLIDENTIFIED BY
connectpw;
EXEC SQL CONNECT TO unix:gaussdb://localhost/connectdb USER connectuser SQLIDENTIFIED BY
"connectpw";
EXEC SQL CONNECT TO unix:gaussdb://localhost/connectdb USER connectuser USING "connectpw";
```

The following is an example of using the connection syntax:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    /* Define host variables by defining columns such as database and password required by the connection
string. The actual values are read from environment variables or configuration files. Environment variables
need to be configured as required. If no environment variable is used, a character string can be directly
assigned. */
    EXEC SQL BEGIN DECLARE SECTION;
    char* db = (char*)malloc((sizeof(char) * 200));
    char* pw = (char*)malloc((sizeof(char) * 200));
    EXEC SQL END DECLARE SECTION;
    char* db_env = getenv("EXAMPLE_DATABASENAME_ENV");
    char* pw_env =  getenv("EXAMPLE_PASSWD_ENV");
    strcpy(db, db_env);
    strcpy(pw, pw_env);

    /* The connection statement involves the database, username, and password. The user must be created
in advance and have related operation permissions. */

    /* Connection mode: EXEC SQL CONNECT TO [ database_name ][ @host ][ :port ] [ USER
connection_user ] */
    /* Case 1: Use the default local connection mode to connect to the gaussdb database. */
    EXEC SQL CONNECT TO gaussdb;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    /* Case 2: Use the default local connection mode to connect to the gaussdb database. The connection alias is conn1. */
    EXEC SQL CONNECT TO gaussdb AS conn1;

    /* Case 3: Use the ip+port mode (localhost indicates the local IP address listened by the database, and $PORT indicates the listening port of the database) to connect to the connectdb database, specify the database alias, and specify the user password. Replace $PORT with the actual port number. */
    EXEC SQL CONNECT TO connectdb@localhost:$PORT AS conn2 USER connectuser USING :pw;

    /* Case 4: Use the ip+port mode (127.0.0.1 indicates the local address listened by the database, and $PORT indicates the listening port of the database) to connect to the connectdb database, specify the database alias, and specify the user password. */
    EXEC SQL CONNECT TO connectdb@127.0.0.1:$PORT AS conn3 USER connectuser SQLIDENTIFIED BY :pw;

    /* Case 5: Close the connection to the database. */
    EXEC SQL DISCONNECT gaussdb;
    EXEC SQL DISCONNECT conn1;
    EXEC SQL DISCONNECT conn2;
    EXEC SQL DISCONNECT conn3;

    /* Connection mode: EXEC SQL CONNECT TO <tcp|unix>:gaussdb://host [ :port ]/[ database_name ][ ? connection_option ] */
    /* Case 1: Replace the URL variables with the host variables pw and db. */
    strcpy(db, "tcp:gaussdb://localhost/connectdb");
    EXEC SQL CONNECT TO :db USER connectuser USING :pw;

    /* Case 2: 127.0.0.1 indicates the IP address listened by the database, and connectdb indicates the database. */
    EXEC SQL CONNECT TO tcp:gaussdb://127.0.0.1/connectdb AS conn4 USER connectuser USING :pw;

    /* Case 3: 127.0.0.1 indicates the IP address listened by the database, connectdb indicates the database, and connect_timeout=14 indicates the connection string configuration parameter. */
    EXEC SQL CONNECT TO tcp:gaussdb://localhost/connectdb?connect_timeout=14 AS conn5 USER connectuser SQLIDENTIFIED BY :pw;

    /* Case 4: Close all connections. */
    EXEC SQL CLOSE ALL;

    /* Connect to the database and execute the service. */
    EXEC SQL CONNECT TO tcp:gaussdb://127.0.0.1/connectdb AS conn4 USER connectuser USING :pw;
    EXEC SQL SET AUTOCOMMIT = ON;
    EXEC SQL CREATE TABLE t1(a INT);
    EXEC SQL INSERT INTO t1 VALUES(1), (2);
    EXEC SQL SELECT a FROM t1 WHERE a > 1;
    EXEC SQL DROP TABLE t1;
    EXEC SQL DISCONNECT CURRENT;
    free(pw);
    free(db);

    return 0;
}
```

The following is an example of using a host variable to specify connection parameters:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    /* The values of dbname, user, and pwd must be read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned. */
    char *dbname     = getenv("EXAMPLE_DBNAME_ENV");    /* Database name */
    char *user       = getenv("EXAMPLE_USERNAME_ENV");  /* Username for connection */
    char *pwd        = getenv("EXAMPLE_PASSWD_ENV");    /* Password */
    char *connection = "tcp:gaussdb://localhost:$PORT/testdb";  /* Connection string. Replace $PORT with
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
the actual port number. */
    char ver[256];                /* Buffer for storing version strings */
    EXEC SQL END DECLARE SECTION;

    /* Enable debug logging. */
    ECPGdebug(1, stderr);

    /* Connect to the database. */
    EXEC SQL CONNECT TO :dbname;
    /* Set the search path. */
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false);
    EXEC SQL COMMIT;
    /* Query the database version. */
    EXEC SQL SELECT version() INTO :ver;
    /* Close the connection. */
    EXEC SQL DISCONNECT;

    /* Print the database version. */
    printf("version: %s\n", ver);

    /* Use the connection string to connect to the database. */
    EXEC SQL CONNECT TO :connection USER :user USING :pwd;
    /* Set the search path. */
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false);
    EXEC SQL COMMIT;
    /* Query the database version. */
    EXEC SQL SELECT version() INTO :ver;
    /* Close the connection. */
    EXEC SQL DISCONNECT;

    /* Print the database version. */
    printf("version: %s\n", ver);

    return 0;
}
```

The running result is as follows (the result varies according to the local environment and is for reference only):

```
version: gaussdb (GaussDB Kernel 506.0.0 build 7cc7ccbb) compiled at 2025-02-24 09:17:26 commit 10340
last mr 22441 debug
version: gaussdb (GaussDB Kernel 506.0.0 build 7cc7ccbb) compiled at 2025-02-24 09:17:26 commit 10340
last mr 22441 debug
```

Helpful Links

DISCONNECT and **SET CONNECTION**

5.8.3.7.3 DEALLOCATE DESCRIPTOR

Description

Deallocates an SQL descriptor area.

Syntax

```
DEALLOCATE DESCRIPTOR name
```

Parameters

name

SQL descriptor name. It is case-sensitive and is an SQL identifier or a host variable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
DEALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

ALLOCATE DESCRIPTOR, **GET DESCRIPTOR**, and **SET DESCRIPTOR**

5.8.3.7.4 DECLARE

Description

Declares a cursor for iterating over the result set of a prepared statement. This command is slightly semantically different from the kernel SQL command **DECLARE**: Whereas the latter executes a query and prepares the result set for retrieval, this embedded SQL command merely declares a name as a "loop variable" for iterating over the result set of a query; the actual execution happens when the cursor is opened with the **OPEN** command.

Syntax

```
DECLARE cursor_name [ BINARY ]  [ NO  SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR prepared_name
DECLARE cursor_name [ BINARY ]  [ NO SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR query
```

Parameters

- **cursor_name**

  Cursor name, which is case-sensitive. It can be an SQL identifier or a host variable.

- **prepared_name**

  Name of the prepared query, which can be an SQL identifier or a host variable.

- **query**

  A SELECT command for providing the rows to be returned by the cursor.

NOTE

For detailed explanations of the cursor options, refer to "SQL Reference > SQL Syntax > D > DECLARE" in *Reference*.

Examples

Examples of declaring a cursor used for query:

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table;
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T;
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version();
```

Example of declaring a cursor for a prepared statement:

```
EXEC SQL PREPARE stmt1 AS SELECT version();
EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

Helpful Links

OPEN

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.7.5 DESCRIBE

Description

Retrieves metadata information for the result columns contained in prepared statements.

Syntax

```
DESCRIBE [ OUTPUT ] prepared_name USING  SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO sqlda_name
```

Parameters

- **prepared_name**

  Prepared statement name. It can be an SQL identifier or a host variable.
- **descriptor_name**

  Descriptor name. It is case-sensitive and is an SQL identifier or a host variable.
- **sqlda_name**

  SQLDA variable name. For details, see **SQLDA**.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME;
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

ALLOCATE DESCRIPTOR and **GET DESCRIPTOR**

5.8.3.7.6 DISCONNECT

Description

Closes one or all database connections.

Syntax

```
DISCONNECT connection_name
DISCONNECT [ CURRENT ]
DISCONNECT DEFAULT
DISCONNECT ALL
```

Parameters

- **connection_name**

  Specifies the name of the database connection established by the **CONNECT** command.
- **CURRENT**

  Closes the current connection, which can be a recently opened connection or a connection set by the **SET CONNECTION** command. If no parameter is

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

passed to the **DISCONNECT** command, the current connection is closed by default.

- **DEFAULT**

  Closes the default connection.

- **ALL**

  Closes all open connections.

Examples

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb AS DEFAULT;
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL CONNECT TO testdb AS con2;
    EXEC SQL CONNECT TO testdb AS con3;
    EXEC SQL DISCONNECT CURRENT;  /* Close connection 3. */
    EXEC SQL DISCONNECT DEFAULT;  /* Close the default connection. */
    EXEC SQL DISCONNECT ALL;      /* Close connections 2 and 1. */
    return 0;
}
```

Helpful Links

CONNECT and **SET CONNECTION**

5.8.3.7.7 EXECUTE IMMEDIATE

Description

Immediately prepares and executes a dynamically specified SQL statement without retrieving result rows.

Syntax

```
EXECUTE IMMEDIATE string
```

Parameters

string

A C string or host variable that contains the SQL statement to be executed.

Examples

The following is an example of executing the INSERT statement using EXECUTE IMMEDIATE and a host variable named **command**:

```
sprintf(command, "INSERT INTO test (name, amount, letter) VALUES ('db: ''r1''', 1, 'f')");
EXEC SQL EXECUTE IMMEDIATE :command;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.7.8 GET DESCRIPTOR

Description

Retrieves information about a query result set and stores it into host variables. A descriptor area is typically populated using FETCH or SELECT before using this command to transfer the information into host language variables. This command can be in either of the following formats:

- Retrieves the header item of a descriptor, which is used to view the result set comprehensively.
- Retrieves information about a particular column, requiring the column number as additional parameter.

Syntax

```
GET DESCRIPTOR descriptor_name VALUE column_number :cvariable = descriptor_item [, ... ]
GET DESCRIPTOR descriptor_name:cvariable = descriptor_header_item [, ... ]
```

Parameters

- **descriptor_name**

  Descriptor name.

- **column_number**

  Number of the column about which information is to be retrieved. The count starts at 1.

- **descriptor_item**

  Information item about a column to be retrieved.

- **cvariable**

  A host variable that will receive the data retrieved from the descriptor area.

- **descriptor_header_item**

  Identifies the header item to be retrieved. Currently, only COUNT that is used to obtain the number of columns in the result set is supported.

Examples

Retrieve the number of columns in a result set.

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
```

Retrieve the data length in the first column.

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
```

Retrieve the data body of the second column as a string.

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA;
```

Run **SELECT current_database();**. The number of columns, column data length, and column data are displayed.

Prerequisites: The **testdb** database has been created.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL BEGIN DECLARE SECTION;
    int  d_count = 0;
    char d_data[1024] = {0};
    int  d_returned_octet_length = 0;
    EXEC SQL END DECLARE SECTION;

    /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false);
    EXEC SQL COMMIT;
    EXEC SQL ALLOCATE DESCRIPTOR d;

    /* Declare and open a cursor, and allocate a descriptor to the cursor. */
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database();
    EXEC SQL OPEN cur;
    EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;

    /* Obtain the total number of columns. */
    EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
    printf("d_count = %d\n", d_count);

    /* Obtain the length of a returned column. */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
    printf("d_returned_octet_length = %d\n", d_returned_octet_length);

    /* Fetch the returned column as a string. */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_data = DATA;
    printf("d_data = %s\n", d_data);

    /* Closed */
    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;

    EXEC SQL DEALLOCATE DESCRIPTOR d;
    EXEC SQL DISCONNECT ALL;

    return 0;
}
```

The result is as follows:

```
d_count                 = 1
d_returned_octet_length = 6
d_data                  = testdb
```

Helpful Links

ALLOCATE DESCRIPTOR, **DEALLOCATE DESCRIPTOR**, and **SET DESCRIPTOR**

5.8.3.7.9 OPEN

Description

Opens a cursor and optionally binds actual values to placeholders in the cursor declaration. The cursor must have been declared using the DECLARE command. Executing the **OPEN** command triggers a query on the server.

Syntax

```
OPEN cursor_name
OPEN cursor_name USING value [, ... ]
OPEN cursor_name USING SQL DESCRIPTOR descriptor_name
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **cursor_name**

  Name of the cursor to be opened. It can be an SQL identifier or a host variable.

- **value**

  A value that is to be bound to a placeholder in the cursor declaration. It can be an SQL constant, a host variable, or a host variable with an indicator.

- **descriptor_name**

  Name of the descriptor that contains the value to be bound to the placeholder in the cursor declaration. It can be an SQL identifier or a host variable.

Examples

```
EXEC SQL OPEN a;
EXEC SQL OPEN d USING 1, 'test';
EXEC SQL OPEN c1 USING SQL DESCRIPTOR mydesc;
EXEC SQL OPEN :curname1;
```

Helpful Links

DECLARE

5.8.3.7.10 PREPARE

Description

Prepares the statement to be executed.

Syntax

```
PREPARE name FROM string
```

Parameters

- **name**

  An identifier for the prepared query.

- **string**

  Dynamically-constructed SQL statement, which can be a C string or a host variable.

Examples

```
char *stmt = "SELECT * FROM test1 WHERE a = ? AND b = ?";
EXEC SQL ALLOCATE DESCRIPTOR outdesc;
EXEC SQL PREPARE foo FROM :stmt;
EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL DESCRIPTOR outdesc;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

The PREPARE statement provided by ECPG is not equivalent to the PREPARE syntax provided by the kernel. The following is an example:

GaussDB syntax:

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

Embedded SQL statement:

```
EXEC SQL PREPARE I(int, int) AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I(1, 2);
```

When the preceding statement is executed, an error message "too few arguments on" is reported.

Example of using dynamic SQL syntax rules to solve the preceding problem:

```
EXEC SQL PREPARE I AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I using 1, 2;
```

5.8.3.7.11 SET AUTOCOMMIT

Description

Sets the autocommit behavior of the current database session. By default, embedded SQL programs do not automatically commit, so you need to explicitly issue COMMIT. This command can change the session to the automatic commit mode so that each individual statement is implicitly committed.

Syntax

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

5.8.3.7.12 SET CONNECTION

Description

Sets a database connection.

Syntax

```
SET CONNECTION { TO | = } connection_name
```

Parameters

connection_name

Specifies the name of the database connection established by the **CONNECT** command.

Examples

```
EXEC SQL SET CONNECTION TO con2;
EXEC SQL SET CONNECTION = con1;
```

Helpful Links

CONNECT and **DISCONNECT**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.8.3.7.13 SET DESCRIPTOR

Description

Populates an SQL descriptor area, which is usually used to bind parameters in a prepared query execution. This command has the following functions:

- Applies to the descriptor header, which is independent of specific data.
- Assigns a value to specific data identified by a number.

Syntax

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ... ]
SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value [, ...]
```

Parameters

- **descriptor_name**

  Descriptor name.

- **descriptor_header_item**

  Identifies the header information item to be set. Currently, only COUNT that can be used to set the number of descriptor items is supported.

- **number**

  Number of descriptor items to be set. The count starts at 1.

- **descriptor_item**

  Identifies which descriptor item is to be set. Currently, only DATA, TYPE, and LENGTH are supported.

- **value**

  Value to be stored in the descriptor item. The value can be an SQL constant or a host variable.

Examples

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1;
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val1, DATA = 'some string';
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA = :val2;
```

Helpful Links

ALLOCATE DESCRIPTOR, **DEALLOCATE DESCRIPTOR**, and **GET DESCRIPTOR**

5.8.3.7.14 TYPE

Description

Defines a new data type. This command is identified only when ECPG is run with the **-c** option.

Syntax

```
TYPE type_name IS ctype
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **type_name**

  Data type name.

- **ctype**

  C language data type.

Examples

```
EXEC SQL TYPE customer IS
   struct
   {
      varchar name[50];
      int     phone;
   };

EXEC SQL TYPE cust_ind IS
   struct ind
   {
      short   name_ind;
      short   phone_ind;
   };

EXEC SQL TYPE c IS char reference;
EXEC SQL TYPE ind IS union { int integer; short smallint; };
EXEC SQL TYPE intarray IS int[AMOUNT];
EXEC SQL TYPE str IS varchar[BUFFERSIZE];
EXEC SQL TYPE string IS char[11];
```

Example of using **EXEC SQL TYPE**:

Prerequisites: The **-c** parameter has been added in the ECPG preprocessing phase, and the **testdb** database has been created in advance.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

EXEC SQL WHENEVER SQLERROR SQLPRINT;
EXEC SQL TYPE tt IS
   struct
   {
      varchar v[256];
      int     i;
   };
EXEC SQL TYPE tt_ind IS
   struct ind {
      short   v_ind;
      short   i_ind;
   };

int main(void)
{
   EXEC SQL BEGIN DECLARE SECTION;
   tt t;
   tt_ind t_ind;
   EXEC SQL END DECLARE SECTION;

   /* Connect to the database. */
   EXEC SQL CONNECT TO testdb AS con1;

   /* Query the current database name along with the fixed value 256, and store the result to the structure
and indicator variable. */
   EXEC SQL SELECT current_database(), 256 INTO :t:t_ind LIMIT 1;

   /* Print the result. */
   printf("t.v = %s\n", t.v.arr);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    printf("t.i = %d\n", t.i);

    printf("t_ind.v_ind = %d\n", t_ind.v_ind);
    printf("t_ind.i_ind = %d\n", t_ind.i_ind);

    /* Disconnect from the database. */
    EXEC SQL DISCONNECT con1;

    return 0;
}
```

The result is as follows:

```
t.v = testdb
t.i = 256
t_ind.v_ind = 0
t_ind.i_ind = 0
```

5.8.3.7.15 VAR

Description

Assigns a new C data type to a host variable. The host variable must have been declared in a DECLARE segment.

NOTE

- Exercise caution when using VAR. Using VAR to change the data type my cause the memory address to be invalid. As a result, the data variable is invalid and the value cannot be assigned.
- If the data type has been defined in the host variable DECLARE segment, you do not need to use the VAR statement.

Syntax

```
VAR varname IS ctype
```

Parameters

- **varname**

  Name of a C variable.

- **ctype**

  C type description.

Examples

```
EXEC SQL BEGIN DECLARE SECTION;
    short a;
EXEC SQL END DECLARE SECTION;
EXEC SQL VAR a IS int;
```

5.8.3.7.16 WHENEVER

Description

Defines an action to be called when an SQL execution exception (row not found, SQL warning, or error) occurs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

Parameters

- **SQLERROR**

  Specifies the action to be called when an error occurs during SQL statement execution.

- **SQLWARNING**

  Specifies the action to be called when a warning occurs during SQL statement execution.

- **NOT FOUND**

  Specifies the action to be called when an SQL statement retrieves or affects zero rows.

NOTE

action indicates the action to be performed. For details, see **Setting Callbacks**.

Examples

```
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLWARNING DO warn();
EXEC SQL WHENEVER SQLERROR sqlprint;
EXEC SQL WHENEVER SQLERROR SQLCALL print2();
EXEC SQL WHENEVER SQLERROR DO handle_error("select");
EXEC SQL WHENEVER SQLERROR DO sqlnotice(NULL, NONO);
EXEC SQL WHENEVER SQLERROR DO sqlprint();
EXEC SQL WHENEVER SQLERROR GOTO error_label;
EXEC SQL WHENEVER SQLERROR STOP;
```

Use WHENEVER NOT FOUND BREAK to process the loop of the result set. The following is a complete example:

Prerequisites: The **testdb** database has been created.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL ALLOCATE DESCRIPTOR d;
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(), 'hoge', 256;
    EXEC SQL OPEN cur;
    EXEC SQL BEGIN DECLARE SECTION;
    char* d1;
    char* d2;
    EXEC SQL END DECLARE SECTION;

    /* When the end of the result set is reached, exit the loop. */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;
        exec sql get descriptor d value 1 :d1=DATA;
        exec sql get descriptor d value 2 :d2=DATA;
        printf("d1 is %s,%s\n", d1, d2) ;
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;
    EXEC SQL DEALLOCATE DESCRIPTOR d;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

The result is as follows:

```
d1 is testdb,hoge
```

5.8.4 ECPG-based Preprocessing, Compilation, and Execution

Prerequisites

You have prepared an embedded SQL-C source file (.pgc). ECPG will convert this file into a standard C program that can be compiled by a C compiler.

The resulting C program is then compiled into an executable using the **gcc** command. Running this executable enables the client program to access databases.

File Inclusion

The following header files are generally required during embedded SQL-C program development:

```
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE sqlca;
```

sqlca refers to the SQL communication area, which stores status information (such as success or error codes) after SQL statement execution. For details about **sqlca**, see **sqlca**.

To include an external file into your embedded SQL program, use the following statements:

```
EXEC SQL INCLUDE filename;
EXEC SQL INCLUDE <filename>;
EXEC SQL INCLUDE "filename";
```

NOTE

- ECPG searches for files in the following order:
  1. Current directory
  2. /usr/local/include
  3. GaussDB directory (defined at compile time)
  4. /usr/include
- However, when EXEC SQL INCLUDE *"filename"* is used, only the current directory is searched.
- In each directory, the preprocessor will first look for the file name as given, and if not found will append .h to the file name and try again (unless the specified file name already has that suffix).
- The file name is case-sensitive.

When using C-style syntax to include custom header files that contain ECPG syntax, refer to these preprocessing examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **a.pgc** file:
  ```
  #include "a.h"
  ```
- **a.h** file:
  ```
  #ifndef _A_H
  #define _A_H
  EXEC SQL BEGIN DECLARE SECTION;
  char sDbName[50] ;
  char sUser[50];
  char sPwd[50];
  EXEC SQL END DECLARE SECTION;
  #endif
  ```
- Compilation methods:
  a. Preprocess via **cpp** commands to expand header files and generate a new .pgc file.
     ```
     cpp -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/gaussdb/server/ -E -C a.pgc a.tmp.d
     cpp -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/gaussdb/server/ -P a.tmp.d a.tmp.pgc
     ```
  b. Preprocess via ECPG.
     ```
     ecpg -I $GAUSSHOME/include/ecpg  -o a.tmp.c a.tmp.pgc
     ```
  c. Compile via GCC.
     ```
     gcc -g -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/gaussdb/server/ -L $GAUSSHOME/lib a.tmp.c -o a -lecpg -lrt -lpq -lpgtypes -lpthread
     ```

NOTE

- When including custom header files via ECPG, do not mix the C-style **#include** directive with the ECPG equivalent. Otherwise, the header files may be expanded repeatedly.
- When working with **cpp** commands, do not directly specify the TCP or Unix protocol in the connection string. Instead, use host variables (refer to **Data Types and Host Variables**) if these protocol types are required.

Conditional Compilation

ECPG provides conditional compilation directives, including **ifdef**, **ifndef**, **else**, **elif**, and **endif**. During preprocessing, different parts of the program are compiled based on different conditions. These directives must be prefixed with **EXEC SQL** when used.

Examples:

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```

The define and undef Directives

Embedded SQL has directives similar to **#define** in the C language:

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
EXEC SQL UNDEF name;
```

Examples:

```
/* Define a name. */
EXEC SQL DEFINE HAVE_FEATURE;

/* Define constants. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';

/* Use UNDEF to remove a definition. */
EXEC SQL UNDEF MYNUMBER;
```

Both standard C-style **#define** and **#undef** and SQL-specific **EXEC SQL DEFINE** can be used in embedded SQL programs. The key distinction lies in when the defined value is evaluated: C-style **#define** is processed by the C preprocessor and undergoes text substitution at compile time. **EXEC SQL DEFINE** is evaluated and substituted during the ECPG preprocessing stage. This means that definitions made by **EXEC SQL DEFINE** take effect before the SQL code is processed, influencing the subsequent generation of SQL statements. In the following example, ECPG performs the substitution, and your compiler will not parse any name or identifier **MYNUMBER**:

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

NOTICE

You cannot use **#define** for a variable that you are going to use in an embedded SQL query because in this case the embedded SQL precompiler is unable to recognize this declaration.

Preprocessing

ECPG-based preprocessing command (using the **test.pgc** file as an example):

```
ecpg -I $GAUSSHOME/include/ecpg -o test.c test.pgc
```

Option setting for ECPG-based preprocessing:

```
ecpg [OPTION]...
```

Supported **OPTION** values include:

- **-o** *OUTFILE*: writes the preprocessing result of the embedded SQL-C program into a C-language file (**test.c**).
- **-I** *DIRECTORY*: specifies the header file search path.
- **-c**: automatically generates a C-language file for the embedded SQL-C program being preprocessed.
- **--version**: displays the current ECPG version.
- **-C** *MODE*: specifies the compatibility mode (for example, **ORA** for Oracle compatibility).
- **-r** *OPTION*: specifies the runtime behavior. Valid values for this *OPTION* include:
  - **no_indicator**: permits using special values to represent null.
  - **prepare**: forces PREPARE before executing all statements.
  - **questionmarks**: permits using question marks (?) as placeholders.
  - **with_hold**: treats cursors created without the HOLD keyword as **WITH HOLD** by default. (The ECPG and kernel versions must match.)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Compilation

To compile the preprocessed embedded SQL-C program, run the following command:

```
gcc -I $GAUSSHOME/include/ecpg -I  $GAUSSHOME/include -I $GAUSSHOME/include/gaussdb/server/ -L $GAUSSHOME/lib test.c -o test  -lecpg -lrt -lpq -lpgtypes -lpthread
```

Execution

Run the following command:

```
./test
```

NOTICE

- As a compilation preprocessing tool, if ECPG reports an error indicating that the header file or function implementation cannot be found during preprocessing or compilation, you can specify the header file or connect to the dynamic library as required.
- Other dynamic libraries and header files on which the ECPG depends are usually stored in the *$GAUSSHOME*/**include/libpq** and *$GAUSSHOME*/**include** directories.
- Common dynamic library dependencies during compilation include: **-lpq**, **-lpq_ce**, and **-lpthread**. If the libpq communication library is required during development, connect to **-lpq** and **-lpq_ce**. If the multi-thread connection is required during development, connect to **-lpthread**.

5.8.5 Typical Application Development Examples

5.8.5.1 Creating a Table, Inserting Data, and Querying Data

Prerequisites: The **testdb** database has been created.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

If **ip-port** is specified in the connection statement during database connection, you must provide *username*/*password*. GaussDB enforces this rule for communication authentication. If **ip-port** is not specified, the local *$PGPORT* (UDS) is used for communication.

The following describes several common methods for connecting to the database. For details, see **Managing Database Connections** and **CONNECT**.

```
/* The connection statement involves the database, username, and password. The user must be created in advance and have related operation permissions. */

/* Connection mode: EXEC SQL CONNECT TO [ database_name ][ @host ][ :port ] [ USER connection_user ] */
/* Case 1: Use the default local connection mode to connect to the gaussdb database. */
EXEC SQL CONNECT TO gaussdb;

/* Case 2: Use the default local connection mode to connect to the gaussdb database. The connection alias is conn1. */
EXEC SQL CONNECT TO gaussdb AS conn1;

/* Case 3: Use the ip+port mode (localhost indicates the local IP address listened by the database, and $PORT indicates the listening port of the database) to connect to the connectdb database, specify the database alias, and specify the user password. Replace $PORT with the actual port number. */
EXEC SQL CONNECT TO connectdb@localhost:$PORT AS conn2 USER connectuser USING :pw;

/* Case 4: Use the ip+port mode (127.0.0.1 indicates the local address listened by the database, and $PORT indicates the listening port of the database) to connect to the connectdb database, specify the database alias, and specify the user password. */
EXEC SQL CONNECT TO connectdb@127.0.0.1:$PORT AS conn3 USER connectuser SQLIDENTIFIED BY :pw;
```

This example demonstrates how to use ECPG to create a table, insert data, and query data.

```
#include <locale.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

EXEC SQL WHENEVER SQLERROR SQLPRINT;
EXEC SQL INCLUDE sqlca;

int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *temp_str = (char *)malloc(11);
    EXEC SQL END DECLARE SECTION;

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb;

    /* Enable automatic commit. You do not need to manually commit the EXEC SQL statement. */
    EXEC SQL SET AUTOCOMMIT = ON;
    EXEC SQL DROP TABLE IF EXISTS test_t;

    /* Create a table and insert data. */
    EXEC SQL CREATE TABLE test_t(f FLOAT, i INT, a INT[10], mstr CHAR(10));
    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(1.01, 1, '{0,1,2,3,4,5,6,7,8,9}', 'China');

    /* Disable the automatic commit. The following SQL statements for inserting data must be manually committed: */
    EXEC SQL SET AUTOCOMMIT = OFF;
    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(2.01, 2, '{0,1,2,3,4,5,6,7,8,9}', 'USA');
    EXEC SQL COMMIT;

    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(3.01, 3, '{0,1,2,3,4,5,6,7,8,9}', 'AUS');
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(4.01, 4, '{0,1,2,3,4,5,6,7,8,9}', 'JAP');
    EXEC SQL COMMIT;

    EXEC SQL BEGIN DECLARE SECTION;
    int a[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
    int id = 6;
    EXEC SQL END DECLARE SECTION;

    /* Fetch data from the host variable and insert the data into the table. The type of the host variable is
the same as that defined in the table. */
    strcpy(temp_str, "RUS");
    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(5.01, 5, :a, :temp_str);
    EXEC SQL COMMIT;

    EXEC SQL SET AUTOCOMMIT = ON;
    EXEC SQL BEGIN;
    EXEC SQL INSERT INTO test_t(f, i, a, mstr) VALUES(6.01, :id, :a, 'SIG');
    EXEC SQL COMMIT;
    EXEC SQL SET AUTOCOMMIT = OFF;

    EXEC SQL BEGIN DECLARE SECTION;
    float ff = 0.0f;
    char tmp_text[25] = "klmnopqrst";
    EXEC SQL END DECLARE SECTION;

    EXEC SQL SET AUTOCOMMIT = ON;
    EXEC SQL BEGIN WORK;

    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    /* Example of a conditional query statement */
    EXEC SQL SELECT f, mstr INTO :ff, :tmp_text FROM test_t WHERE f > (SELECT f FROM test_t WHERE i = 4
OR i < 0) ORDER BY a LIMIT 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    EXEC SQL SELECT f, mstr INTO :ff, :tmp_text FROM test_t WHERE mstr = 'JAP' ORDER BY i;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    EXEC SQL SELECT f, mstr INTO :ff, :tmp_text FROM test_t ORDER BY i DESC LIMIT 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    EXEC SQL SELECT f, mstr INTO :ff, :tmp_text FROM test_t ORDER BY mstr LIMIT 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    EXEC SQL SELECT COUNT(f), a INTO :ff, :tmp_text FROM test_t WHERE i > 2 GROUP BY a LIMIT 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    EXEC SQL SELECT COUNT(f), a INTO :ff, :tmp_text FROM test_t WHERE i > 3 GROUP BY a ORDER BY a
LIMIT 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    EXEC SQL SELECT SUM(f), a INTO :ff, :tmp_text FROM test_t WHERE i > 2 GROUP BY a ORDER BY a
LIMIT 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    EXEC SQL SELECT DISTINCT a INTO :tmp_text FROM test_t ORDER BY a LIMIT 1;

    EXEC SQL DROP TABLE test_t;

    EXEC SQL COMMIT;
    /* Release the connection and release the memory allocated to the host variable. */
    EXEC SQL DISCONNECT;
    free(temp_str);

    return 0;
}
```

The result is as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Found ff=0.000000 tmp_text=klmnopqrst
Found ff=5.010000 tmp_text=RUS
Found ff=4.010000 tmp_text=JAP
Found ff=6.010000 tmp_text=SIG
Found ff=3.010000 tmp_text=AUS
Found ff=2.000000 tmp_text={9,8,7,6,5,4,3,2,1,0}
Found ff=1.000000 tmp_text={0,1,2,3,4,5,6,7,8,9}
Found ff=7.020000 tmp_text={0,1,2,3,4,5,6,7,8,9}
```

5.8.5.2 Date, Timestamp, and Interval Parsing

This example uses a library function to perform different operations on time and date types. For details, see **Using Library Functions**.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include <pgtypes_date.h>
#include <pgtypes_timestamp.h>

char *dates[] = { "19990108foobar",
                  "19990108 foobar",
                  "1999-01-08 foobar",
                  "January 8, 1999",
                  "1999-01-08",
                  "1/8/1999",
                  "1/18/1999",
                  "01/02/03",
                  "1999-Jan-08",
                  "Jan-08-1999",
                  "08-Jan-1999",
                  "99-Jan-08",
                  "08-Jan-99",
                  "08-Jan-06",
                  "Jan-08-99",
                  "19990108",
                  "990108",
                  "1999.008",
                  "J2451187",
                  "January 8, 99 BC",
                  NULL
                 };

/* The value cannot conflict with the value of times of libc. */
static char *times[] = { "0:04",
                         "1:59 PDT",
                         "13:24:40 -8:00",
                         "13:24:40.495+3",
                         NULL
                        };

char *intervals[] = { "1 minute",
                      "1 12:59:10",
                      "2 day 12 hour 59 minute 10 second",
                      "1 days 12 hrs 59 mins 10 secs",
                      "1 days 1 hours 1 minutes 1 seconds",
                      "1 year 59 mins",
                      "1 year 59 mins foobar",
                      NULL
                    };

int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
        date date1;
        timestamp ts1, ts2;
        char *text;
        interval *i1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    date *dc;
EXEC SQL END DECLARE SECTION;

int i, j;
char *endptr;


/* Parse a timestamp from its textual representation and convert the date into a character string. */
ts1 = PGTYPEStimestamp_from_asc("2003-12-04 17:34:29", NULL);
text = PGTYPEStimestamp_to_asc(ts1);

printf("timestamp: %s\n", text);
free(text);

/* Extract the date part from the timestamp. */
date1 = PGTYPESdate_from_timestamp(ts1);
dc = PGTYPESdate_new();
*dc = date1;
/* Return the textual representation of a date variable. */
text = PGTYPESdate_to_asc(*dc);
printf("Date of timestamp: %s\n", text);
free(text);
PGTYPESdate_free(dc);

for (i = 0; dates[i]; i++)
{
    bool err = false;
    /* Parse a date from the textual representation of the date. */
    date1 = PGTYPESdate_from_asc(dates[i], &endptr);
    if (date1 == INT_MIN) {
        err = true;
    }
    /* Return the textual representation of a date variable. */
    text = PGTYPESdate_to_asc(date1);
    printf("Date[%d]: %s (%c - %c)\n",
           i, err ? "-" : text,
           endptr ? 'N' : 'Y',
           err ? 'T' : 'F');
    free(text);
    if (!err)
    {
        for (j = 0; times[j]; j++)
        {
            int length = strlen(dates[i]) + 1 + strlen(times[j]) + 1;
            char* t = (char *)malloc(length);
            sprintf(t, "%s %s", dates[i], times[j]);
            /* Parse a timestamp from its textual representation and convert the date into a character string.
*/
            ts1 = PGTYPEStimestamp_from_asc(t, NULL);
            text = PGTYPEStimestamp_to_asc(ts1);
            if (i != 19 || j != 3)
                printf("TS[%d,%d]: %s\n", i, j, errno ? "-" : text);
            free(text);
            free(t);
        }
    }
}

/* Parse a timestamp from its textual representation. */
ts1 = PGTYPEStimestamp_from_asc("2004-04-04 23:23:23", NULL);

for (i = 0; intervals[i]; i++)
{
    interval *ic;
    /* Parse an interval from its textual representation. */
    i1 = PGTYPESinterval_from_asc(intervals[i], &endptr);
    if (*endptr)
        printf("endptr set to %s\n", endptr);
    if (!i1)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    {
        printf("Error parsing interval %d\n", i);
        continue;
     }
    /* Add an interval variable to the timestamp variable. */
     j = PGTYPEStimestamp_add_interval(&ts1, i1, &ts2);
     if (j < 0)
        continue;
    /* Convert a variable of the interval type to the text format. */
     text = PGTYPESinterval_to_asc(i1);
     printf("interval[%d]: %s\n", i, text ? text : "-");
     free(text);

    /* Return a pointer to an allocated interval variable. */
     ic = PGTYPESinterval_new();
     /* Copy a variable of the interval type. */
     PGTYPESinterval_copy(i1, ic);
     /* Convert a variable of the interval type to the text format. */
     text = PGTYPESinterval_to_asc(i1);
     printf("interval_copy[%d]: %s\n", i, text ? text : "-");
     free(text);
     /* Release the memory that has been allocated to an interval variable. */
     PGTYPESinterval_free(ic);
     PGTYPESinterval_free(i1);
  }

  return (0);
}
```

The result is as follows:

```
Date of timestamp: 2003-12-04
Date[0]: - (N - T)
Date[1]: - (N - T)
Date[2]: - (N - T)
Date[3]: 1999-01-08 (N - F)
TS[3,0]: 1999-01-08 00:04:00
TS[3,1]: 1999-01-08 01:59:00
TS[3,2]: 1999-01-08 13:24:40
TS[3,3]: 1999-01-08 13:24:40.495
Date[4]: 1999-01-08 (N - F)
TS[4,0]: 1999-01-08 00:04:00
TS[4,1]: 1999-01-08 01:59:00
TS[4,2]: 1999-01-08 13:24:40
TS[4,3]: 1999-01-08 13:24:40.495
Date[5]: 1999-01-08 (N - F)
TS[5,0]: 1999-01-08 00:04:00
TS[5,1]: 1999-01-08 01:59:00
TS[5,2]: 1999-01-08 13:24:40
TS[5,3]: 1999-01-08 13:24:40.495
Date[6]: 1999-01-18 (N - F)
TS[6,0]: 1999-01-18 00:04:00
TS[6,1]: 1999-01-18 01:59:00
TS[6,2]: 1999-01-18 13:24:40
TS[6,3]: 1999-01-18 13:24:40.495
Date[7]: 2003-01-02 (N - F)
TS[7,0]: 2003-01-02 00:04:00
TS[7,1]: 2003-01-02 01:59:00
TS[7,2]: 2003-01-02 13:24:40
TS[7,3]: 2003-01-02 13:24:40.495
Date[8]: 1999-01-08 (N - F)
TS[8,0]: 1999-01-08 00:04:00
TS[8,1]: 1999-01-08 01:59:00
TS[8,2]: 1999-01-08 13:24:40
TS[8,3]: 1999-01-08 13:24:40.495
Date[9]: 1999-01-08 (N - F)
TS[9,0]: 1999-01-08 00:04:00
TS[9,1]: 1999-01-08 01:59:00
TS[9,2]: 1999-01-08 13:24:40
TS[9,3]: 1999-01-08 13:24:40.495
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Date[10]: 1999-01-08 (N - F)
TS[10,0]: 1999-01-08 00:04:00
TS[10,1]: 1999-01-08 01:59:00
TS[10,2]: 1999-01-08 13:24:40
TS[10,3]: 1999-01-08 13:24:40.495
Date[11]: 1999-01-08 (N - F)
TS[11,0]: 1999-01-08 00:04:00
TS[11,1]: 1999-01-08 01:59:00
TS[11,2]: 1999-01-08 13:24:40
TS[11,3]: 1999-01-08 13:24:40.495
Date[12]: 1999-01-08 (N - F)
TS[12,0]: 1999-01-08 00:04:00
TS[12,1]: 1999-01-08 01:59:00
TS[12,2]: 1999-01-08 13:24:40
TS[12,3]: 1999-01-08 13:24:40.495
Date[13]: 2006-01-08 (N - F)
TS[13,0]: 2006-01-08 00:04:00
TS[13,1]: 2006-01-08 01:59:00
TS[13,2]: 2006-01-08 13:24:40
TS[13,3]: 2006-01-08 13:24:40.495
Date[14]: 1999-01-08 (N - F)
TS[14,0]: 1999-01-08 00:04:00
TS[14,1]: 1999-01-08 01:59:00
TS[14,2]: 1999-01-08 13:24:40
TS[14,3]: 1999-01-08 13:24:40.495
Date[15]: 1999-01-08 (N - F)
TS[15,0]: 1999-01-08 00:04:00
TS[15,1]: 1999-01-08 01:59:00
TS[15,2]: 1999-01-08 13:24:40
TS[15,3]: 1999-01-08 13:24:40.495
Date[16]: 1999-01-08 (N - F)
TS[16,0]: 1999-01-08 00:04:00
TS[16,1]: 1999-01-08 01:59:00
TS[16,2]: 1999-01-08 13:24:40
TS[16,3]: 1999-01-08 13:24:40.495
Date[17]: 1999-01-08 (N - F)
TS[17,0]: 1999-01-08 00:04:00
TS[17,1]: 1999-01-08 01:59:00
TS[17,2]: 1999-01-08 13:24:40
TS[17,3]: 1999-01-08 13:24:40.495
Date[18]: 1999-01-08 (N - F)
TS[18,0]: 1999-01-08 00:04:00
TS[18,1]: 1999-01-08 01:59:00
TS[18,2]: 1999-01-08 13:24:40
TS[18,3]: 1999-01-08 13:24:40.495
Date[19]: 0099-01-08 BC (N - F)
TS[19,0]: 0099-01-08 00:04:00 BC
TS[19,1]: 0099-01-08 01:59:00 BC
TS[19,2]: 0099-01-08 13:24:40 BC
interval[0]: @ 1 min
interval_copy[0]: @ 1 min
interval[1]: @ 1 day 12 hours 59 mins 10 secs
interval_copy[1]: @ 1 day 12 hours 59 mins 10 secs
interval[2]: @ 2 days 12 hours 59 mins 10 secs
interval_copy[2]: @ 2 days 12 hours 59 mins 10 secs
interval[3]: @ 1 day 12 hours 59 mins 10 secs
interval_copy[3]: @ 1 day 12 hours 59 mins 10 secs
interval[4]: @ 1 day 1 hour 1 min 1 sec
interval_copy[4]: @ 1 day 1 hour 1 min 1 sec
interval[5]: @ 1 year 59 mins
interval_copy[5]: @ 1 year 59 mins
Error parsing interval 6
```

5.8.5.3 Parsing and Calculating Numeric Types

This example uses the pgtypes library function to perform different operations on the numeric type.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
#include <pgtypes_error.h>
#include <decimal.h>

char* nums[] = { "2E394", "-2", ".794", "3.44", "592.49E21", "-32.84e4",
                 "2E-394", ".1E-2", "+.0", "-592.49E-07", "+32.84e-4",
                 ".500001", "-.5000001",
                 "12345678901234567890123456789.91", /* A 30-bit number should be converted into a decimal
number. */
                 "12345678901234567890123456789.921", /* A 31-bit number cannot be converted into a
decimal number. */
                 "not a number",
                 NULL
                };

static void check_errno(void);

int main(void)
{
    char *text="error\n";
    char *endptr;
    numeric *num, *nin;
    decimal *dec;
    long l;
    int i, j, k, q, r, count = 0;
    double d;
    numeric **numarr = (numeric **) calloc(1, sizeof(numeric));


    for (i = 0; nums[i]; i++)
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nums[i]. */
        num = PGTYPESnumeric_from_asc(nums[i], &endptr);
        if (!num) check_errno();
        if (endptr != NULL)
        {
            printf("endptr of %d is not NULL\n", i);
            if (*endptr != '\0')
                printf("*endptr of %d is not \\0\n", i);
        }
        if (!num) continue;

        numarr = (numeric **)realloc(numarr, sizeof(numeric *) * (count + 1));
        numarr[count++] = num;

        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type num. */
        text = PGTYPESnumeric_to_asc(num, -1);
        if (!text) check_errno();
        printf("num[%d,1]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 0);
        if (!text) check_errno();
         printf("num[%d,2]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 1);
        if (!text) check_errno();
         printf("num[%d,3]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 2);
        if (!text) check_errno();
        printf("num[%d,4]: %s\n", i, text); free(text);

        /* Request a pointer to a newly allocated numeric variable. */
        nin = PGTYPESnumeric_new();
        text = PGTYPESnumeric_to_asc(nin, 2);
        if (!text) check_errno();
        printf("num[%d,5]: %s\n", i, text); free(text);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        /* Convert a numeric variable into a long int variable. */
        r = PGTYPESnumeric_to_long(num, &l);
        if (r) check_errno();
        printf("num[%d,6]: %ld (r: %d)\n", i, r?0L:l, r);
        if (r == 0)
        {
            /* Convert a long int variable into a numeric variable. */
            r = PGTYPESnumeric_from_long(l, nin);
            if (r) check_errno();
            /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
            text = PGTYPESnumeric_to_asc(nin, 2);
            /* Compare two numeric variables. */
            q = PGTYPESnumeric_cmp(num, nin);
            printf("num[%d,7]: %s (r: %d - cmp: %d)\n", i, text, r, q);
            free(text);
        }

        /* Convert a numeric variable into an int variable. */
        r = PGTYPESnumeric_to_int(num, &k);
        if (r) check_errno();
        printf("num[%d,8]: %d (r: %d)\n", i, r?0:k, r);
        if (r == 0)
        {
            /* Convert an int variable into a numeric variable. */
            r = PGTYPESnumeric_from_int(k, nin);
            if (r) check_errno();
            /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
            text = PGTYPESnumeric_to_asc(nin, 2);
            q = PGTYPESnumeric_cmp(num, nin);
            printf("num[%d,9]: %s (r: %d - cmp: %d)\n", i, text, r, q);
            free(text);
        }

        if (i != 6)
        {
            /* Convert a variable of the numeric type to the double-precision type. */
            r = PGTYPESnumeric_to_double(num, &d);
            if (r) check_errno();
            printf("num[%d,10]: %g (r: %d)\n", i, r?0.0:d, r);
        }

       /* Request a pointer to a newly allocated numeric variable. */
        dec = PGTYPESdecimal_new();
        /* Convert a decimal variable into a numeric variable. */
        r = PGTYPESnumeric_to_decimal(num, dec);
        if (r) check_errno();
        printf("num[%d,11]: - (r: %d)\n", i, r);
        if (r == 0)
        {
            /* Convert a decimal variable into a numeric variable. */
            r = PGTYPESnumeric_from_decimal(dec, nin);
            if (r) check_errno();
           /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
            text = PGTYPESnumeric_to_asc(nin, 2);
           /* Compare two numeric variables. */
            q = PGTYPESnumeric_cmp(num, nin);
            printf("num[%d,12]: %s (r: %d - cmp: %d)\n", i, text, r, q);
            free(text);
        }

        /* Release the memory allocated to the numeric variable. */
        PGTYPESdecimal_free(dec);
        PGTYPESnumeric_free(nin);
        printf("\n");
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    for (i = 0; i < count; i++)
    {
        for (j = 0; j < count; j++)
        {
            /* Request a pointer to a newly allocated numeric variable. */
            numeric* a = PGTYPESnumeric_new();
            numeric* s = PGTYPESnumeric_new();
            numeric* m = PGTYPESnumeric_new();
            numeric* d = PGTYPESnumeric_new();
            /* Add two numeric variables to the third numeric variable. */
            r = PGTYPESnumeric_add(numarr[i], numarr[j], a);
            if (r)
            {
              check_errno();
              printf("r: %d\n", r);
             }
            else
            {
                /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type a. */
                text = PGTYPESnumeric_to_asc(a, 10);
                printf("num[a,%d,%d]: %s\n", i, j, text);
                free(text);
            }
            /* Subtract two numeric variables and returns the result to the third numeric variable. */
            r = PGTYPESnumeric_sub(numarr[i], numarr[j], s);
            if (r)
            {
                check_errno();
                printf("r: %d\n", r);
            }
            else
            {
                /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type s. */
                text = PGTYPESnumeric_to_asc(s, 10);
                printf("num[s,%d,%d]: %s\n", i, j, text);
                free(text);
            }
            /* Multiply two numeric variables and returns the result to the third numeric variable. */
            r = PGTYPESnumeric_mul(numarr[i], numarr[j], m);
            if (r)
            {
                check_errno();
                printf("r: %d\n", r);
            }
            else
            {
                /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type m. */
                text = PGTYPESnumeric_to_asc(m, 10);
                printf("num[m,%d,%d]: %s\n", i, j, text);
                free(text);
            }
            /* Divide two numeric variables and returns the result to the third numeric variable. */
            r = PGTYPESnumeric_div(numarr[i], numarr[j], d);
            if (r)
            {
                check_errno();
                printf("r: %d\n", r);
            }
            else
            {
                /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type d. */
                text = PGTYPESnumeric_to_asc(d, 10);
                printf("num[d,%d,%d]: %s\n", i, j, text);
                free(text);
            }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            /* Release the memory allocated to the numeric variable. */
            PGTYPESnumeric_free(a);
            PGTYPESnumeric_free(s);
            PGTYPESnumeric_free(m);
            PGTYPESnumeric_free(d);
        }
    }

    for (i = 0; i < count; i++)
    {
         /* Return a pointer to a string allocated by malloc that contains the string representation of the 
numeric type numarr[i]. */
        text = PGTYPESnumeric_to_asc(numarr[i], -1);
        printf("%d: %s\n", i, text);
        free(text);
        /* Free memory. */
        PGTYPESnumeric_free(numarr[i]);
    }
    free(numarr);

    return (0);
}

/* Error handling */
static void
check_errno(void)
{
    switch(errno)
    {
        case 0:
        printf("(no errno set) - ");
        break;
        case PGTYPES_NUM_OVERFLOW:
        printf("(errno == PGTYPES_NUM_OVERFLOW) - ");
        break;
        case PGTYPES_NUM_UNDERFLOW:
        printf("(errno == PGTYPES_NUM_UNDERFLOW) - ");
        break;
        case PGTYPES_NUM_BAD_NUMERIC:
        printf("(errno == PGTYPES_NUM_BAD_NUMERIC) - ");
        break;
        case PGTYPES_NUM_DIVIDE_ZERO:
        printf("(errno == PGTYPES_NUM_DIVIDE_ZERO) - ");
        break;
        default:
        printf("(unknown errno (%d))\n", errno);
        printf("(libc: (%s)) ", strerror(errno));
        break;
    }
}
```

Running result (only a part of the running result is displayed):

```
endptr of 0 is not NULL
num[0,1]:
2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000
num[0,2]:
2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000
num[0,3]:
2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000.0
num[0,4]:
2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000.00
num[0,5]: 0.00
(errno == PGTYPES_NUM_OVERFLOW) - num[0,6]: 0 (r: -1)
(errno == PGTYPES_NUM_OVERFLOW) - num[0,8]: 0 (r: -1)
(errno == PGTYPES_NUM_OVERFLOW) - num[0,10]: 0 (r: -1)
num[0,11]: - (r: 0)
num[0,12]:
2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000.00 (r: 0 - cmp: 0)

endptr of 1 is not NULL
num[1,1]: -2
num[1,2]: -2
num[1,3]: -2.0
num[1,4]: -2.00
num[1,5]: 0.00
num[1,6]: -2 (r: 0)
num[1,7]: -2.00 (r: 0 - cmp: 0)
num[1,8]: -2 (r: 0)
num[1,9]: -2.00 (r: 0 - cmp: 0)
num[1,10]: -2 (r: 0)
num[1,11]: - (r: 0)
num[1,12]: -2.00 (r: 0 - cmp: 0)

endptr of 2 is not NULL
num[2,1]: 0.794
num[2,2]: 1
num[2,3]: 0.8
num[2,4]: 0.79
num[2,5]: 0.00
num[2,6]: 1 (r: 0)
num[2,7]: 1.00 (r: 0 - cmp: -1)
num[2,8]: 1 (r: 0)
num[2,9]: 1.00 (r: 0 - cmp: -1)
num[2,10]: 0.794 (r: 0)
num[2,11]: - (r: 0)
num[2,12]: 0.79 (r: 0 - cmp: 0)

endptr of 3 is not NULL
num[3,1]: 3.44
num[3,2]: 3
num[3,3]: 3.4
num[3,4]: 3.44
num[3,5]: 0.00
num[3,6]: 3 (r: 0)
num[3,7]: 3.00 (r: 0 - cmp: 1)
num[3,8]: 3 (r: 0)
num[3,9]: 3.00 (r: 0 - cmp: 1)
num[3,10]: 3.44 (r: 0)
num[3,11]: - (r: 0)
num[3,12]: 3.44 (r: 0 - cmp: 0)

endptr of 4 is not NULL
num[4,1]: 59249000000000000000000
num[4,2]: 59249000000000000000000
num[4,3]: 59249000000000000000000.0
num[4,4]: 59249000000000000000000.00
num[4,5]: 0.00
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(errno == PGTYPES_NUM_OVERFLOW) - num[4,6]: 0 (r: -1)
(errno == PGTYPES_NUM_OVERFLOW) - num[4,8]: 0 (r: -1)
num[4,10]: 5.9249e+23 (r: 0)
num[4,11]: - (r: 0)
num[4,12]: 59249000000000000000000.00 (r: 0 - cmp: 0)
... (omitted)
```

5.8.6 ECPG API Reference

5.8.6.1 Overview

The ECPG API reference describes the data type-related APIs provided by the pgtypes library for users to use in embedded SQL-C programs.

The pgtypes library maps SQL data types to C data types and provides some APIs to implement basic functions and calculations.

5.8.6.2 Interval Type

Table 5-132 lists the common APIs for interval data provided by the ECPG.

Table 5-132 Common interval type APIs

API	Description	Remarks
interval* PGTYPESinterval_new(void)	Returns a pointer to a newly allocated interval variable.	This function creates an interval variable on the heap. The return value is of the interval* type.
void PGTYPESinterval_free(interval *inval)	Releases the memory of the allocated interval variable.	This function releases the memory allocated to the interval* variable created by the PGTYPESinterval_new function.
interval* PGTYPESinterval_from_asc(char *str, char **endptr)	Parses an interval from its textual representation.	This function parses the input string **str** and returns a pointer to an allocated interval variable. ECPG can parse complete character strings. Currently, the address of the first invalid character cannot be stored in ***endptr**. You can set **endptr** to **NULL**.
char* PGTYPESinterval_to_asc(interval *span)	Converts a variable of the interval type into its textual representation.	This function converts the interval variable to which the span points into a char*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESinterval_copy(interval *intvlsrc, interval *intvldest)	Copies a variable of the interval type.	This function copies the interval variable that intvlsrc points to into the variable that intvldest points to.

Examples

For details, see **Typical Application Development Examples**.

5.8.6.3 Numeric Type

Table 5-133 lists the common APIs for numeric or decimal data provided by ECPG.

Table 5-133 Common numeric type APIs

API	Description	Remarks
numeric* PGTYPESnumeric_new(void)	Requests a pointer to a newly allocated numeric variable.	This function creates a numeric* variable on the heap. The return value is of the numeric* type.
decimal* PGTYPESdecimal_new(void)	Requests a pointer to a newly allocated decimal variable.	This function creates a decimal* variable on the heap. The return value is of the decimal* type.
void PGTYPESnumeric_free(numeric *var)	Releases the memory of the allocated numeric* variable.	This function frees a numeric* variable created by using the PGTYPESnumeric_new function.
void PGTYPESdecimal_free(decimal *dst)	Releases the memory of the allocated decimal* variable.	This function frees a decimal* variable created by using the PGTYPESdecimal_new function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
numeric* PGTYPESnumeric_from_asc(char *str, char **endptr)	Converts a string to a numeric value.	The valid format is -2, .794, +3.44, 592.49E07, or -32.84e-4. If the parsing and conversion are successful, a valid pointer is returned. Otherwise, a null pointer is returned. ECPG can parse complete character strings. Currently, the address of the first invalid character cannot be stored in ***endptr**. You can set **endptr** to **NULL**.
char* PGTYPESnumeric_to_asc(numeric *num, int dscale)	Converts a numeric variable to a string.	The numeric value will be printed using dscale decimal places and will be rounded if necessary.
int PGTYPESnumeric_add(numeric *var1, numeric *var2, numeric *result)	Adds two numeric variables to the third numeric variable.	This function adds the variables *var1* and *var2* to the result variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_sub(numeric *var1, numeric *var2, numeric *result)	Subtracts two numeric variables and returns the result to the third numeric variable.	This function subtracts *var2* from *var1*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_mul(numeric *var1, numeric *var2, numeric *result)	Multiplies two numeric variables and returns the result to the third numeric variable.	This function multiplies variables *var1* and *var2*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESnumeric_div(numeric *var1, numeric *var2, numeric *result)	Divides two numeric variables and returns the result to the third numeric variable.	This function divides variable *var1* by variable *var2*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_cmp(numeric *var1, numeric *var2)	Compares two numeric variables.	This function compares two numeric variables. If an error is reported, *INT_MAX* is returned. On success, this function returns one of the following three possible results: • If *var1* is greater than *var2*, **1** is returned. • If *var1* is smaller than *var2*, **–1** is returned. • If *var1* and *var2* are equal, **0** is returned.
int PGTYPESnumeric_from_int(signed int int_val, numeric *var)	Converts an integer variable to a numeric variable.	This function accepts a signed int variable and stores it in the numeric variable *var*. It returns **0** on success and **–1** in case of failure.
int PGTYPESnumeric_from_long(signed long int long_val, numeric *var)	Converts a long integer variable to a numeric variable.	This function accepts a long int variable and stores it in the numeric variable *var*. It returns **0** on success and **–1** in case of failure.
int PGTYPESnumeric_copy(numeric *src, numeric *dst)	Copies a numeric variable to another numeric variable.	This function copies the value of the variable that **src** points to into the variable that **dst** points to. The function returns **0** on success and **–1** in case of error.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESnumeric_from_double(double d, numeric *dst)	Converts a double-precision variable to a numeric variable.	This function accepts a double-precision variable and stores the result in the variable that **dst** points to. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_to_double(numeric *nv, double *dp)	Converts a numeric variable into a double-precision variable.	This function converts the numeric value in the variable that **nv** points to into a double-precision variable that **dp** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.
int PGTYPESnumeric_to_int(numeric *nv, int *ip)	Converts a numeric variable into an integer.	This function converts the numeric value in the variable that **nv** points to into an integer variable that **ip** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESnumeric_to_long(numeric *nv, long *ip)	Converts a numeric variable into a long int variable.	This function converts the numeric value in the variable that **nv** points to into a long integer variable that **ip** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.
int PGTYPESnumeric_to_decimal(numeric *src, decimal *dst)	Converts a numeric variable into a decimal variable.	This function converts the numeric value in the variable that **src** points to into a decimal variable that **dst** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.
int PGTYPESnumeric_from_decimal(decimal *src, numeric *dst)	Converts a decimal variable into a numeric variable.	This function converts the decimal value in the variable that **src** points to into a numeric variable that **dst** points to. The function returns **0** on success and **–1** in case of error (or overflow).

Examples

For details, see **Typical Application Development Examples**.

5.8.6.4 Date Type

Table 5-134 lists the common date type APIs provided by ECPG.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-134 Common date type APIs

API	Description	Remarks
date* PGTYPESdate_new(void)	Requests a pointer to a newly allocated date variable.	This function creates a date* variable on the heap. The return value is of the date* type.
void PGTYPESdate_free(date *d)	Releases the memory of the allocated date* variable.	This function frees a date* variable created by using the PGTYPESdate_new function.
date PGTYPESdate_from_asc(char *str, char **endptr)	Parses a date from its textual representation.	This function receives a string **str** and a pointer **endptr** pointing to the string, and converts the string to a date. Currently, you cannot store the address of the first invalid character in ***endptr**. However, you can set **endptr** to **NULL**. **NOTICE** The function always assumes MDY-formatted dates and there is currently no variable to change that within ECPG.
char* PGTYPESdate_to_asc(date dDate)	Returns the textual representation of a date variable.	This function accepts the date **dDate** as its unique parameter. It will output the date in the YYYY-MM-DD format.
date PGTYPESdate_from_timestamp(timestamp dt)	Extracts the date part from a timestamp.	This function accepts a timestamp as its unique parameter and returns the extracted date part from the timestamp.
void PGTYPESdate_julmdy(date jd, int *mdy)	Extracts the values of day, month, and year from a variable of the date type.	This function accepts the date **jd** and a pointer to an array of three integer values **mdy**. The variable name indicates the sequence. **mdy[0]** indicates the month, **mdy[1]** indicates the day, and **mdy[2]** indicates the year.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
void PGTYPESdate_mdyjul(int *mdy, date *jdate)	Creates a date using the specified integer value.	This function accepts an array (**mdy**) consisting of three integers as its first parameter. The three integers are used to indicate the day, month, and year, respectively. The second parameter is a pointer to a variable of the date type, which is used to store the result of the operation.
int PGTYPESdate_dayofweek(date d)	Accepts a date variable and returns an integer indicating the day of the week.	This function accepts the date variable **d** as its unique parameter and returns an integer indicating the day of the week. • **0**: Sunday • **1**: Monday • **2**: Tuesday • **3**: Wednesday • **4**: Thursday • **5**: Friday • **6**: Saturday
void PGTYPESdate_today(date *d)	Obtains the current date.	This function is used to obtain the current date and store it in a date variable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESdate_defmt_asc(date *d, const char *fmt, char *str)	Converts a char* string to a date using a format mask.	This function accepts a pointer **d** pointing to a date value for storing the operation result, a format mask **fmt** for parsing the date, and a char* string **str** containing the textual representation of the date. The textual representation is expected to match the format mask. However, you do not need to have a 1:1 mapping of the string to the format mask. The function only analyzes the sequential order to identify **yy** or **yyyy** (year), **mm** (month), and **dd** (day).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPESdate_fmt_asc(date dDate, const char *fmtstring, char *outbuf)	Converts a variable of the date type into its textual representation using a format mask.	This function accepts the date **dDate** to be converted, the format mask **fmtstring**, and the string **outbuf** of the textual representation of the date to be saved. It returns **0** on success and a negative value in case of error. The following is an example of valid input: 112359 //mmddyy 59/11/23 //yy/mm/dd Nov.23,1959 //mmm.dd,yyyy Mon,Nov.23,1959 // ddd,mmm.dd,yyyy Format: • **dd**: indicates the day of a month. • **mm**: indicates the month of a year. • **yy**: indicates a two-digit year. • **yyyy**: indicates a four-digit year. • **ddd**: indicates the day of the week (abbreviation). • **mmm**: indicates the month (abbreviation).

Examples

For details, see **Typical Application Development Examples**.

5.8.6.5 Timestamp Type

Table 5-135 lists the common APIs for timestamp data provided by ECPG.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-135 Common timestamp type APIs

API	Description	Remarks
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr)	Parses a timestamp from its textual representation into a timestamp variable.	This function accepts a string **str** and a pointer **endptr** pointing to the string. It returns the parsed timestamp on success and **PGTYPESInvalidTimestamp** in case of error. In addition, **errno** is set to **PGTYPES_TS_BAD_TIMESTAMP** when an error occurs. The valid input of **PGTYPEStimestamp_from_asc** is as follows: 1999-01-08 04:05:06 January 8 04:05:06 1999 PST 1999-Jan-08 04:05:06.789-8 1999-01-08 04:05:06.789 (time zone specifier ignored) J2451187 04:05-08:00 1999-01-08 04:05:00 (time zone specifier ignored)
char *PGTYPEStimestamp_to_asc(timestamp tstamp)	Converts a timestamp to a char* string.	This function accepts the timestamp **tstamp** as its unique parameter and returns an allocated string containing the textual representation of the timestamp. The result must be released using PGTYPESchar_free().
void PGTYPEStimestamp_current(timestamp *ts)	Returns the current timestamp.	This function obtains the current timestamp and saves it to the timestamp variable that **ts** points to.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPEStimestamp_fmt_asc(timestamp *ts, char *output, int str_len, char *fmtstr)	Converts a timestamp variable into a char* string using a format mask.	This function accepts a pointer **ts** pointing to the timestamp to be converted, a pointer **output** pointing to the output buffer, the maximum length **str_len** allocated to the output buffer, and the format mask **fmtstr** for conversion as its parameters. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_sub(timestamp *ts1, timestamp *ts2, interval *iv)	Subtracts a timestamp from another timestamp and saves the result in an interval variable.	This function subtracts the timestamp variable that **ts2** points to from the timestamp variable that **ts1** points to, and stores the result in the interval variable that **iv** points to. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_defmt_asc(char *str, char *fmt, timestamp *d)	Parses a string into a timestamp based on the specified format.	This function accepts a timestamp textual representation placed in the variable **str** and a format mask placed in the variable **fmt** that will be used for conversion. The result will be stored in the variable that **d** points to. If the format mask **fmt** is null, the function uses the default format mask **%Y-%m-%d %H:%M:%S**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Remarks
int PGTYPEStimestamp_add_interval(timestamp *tin, interval *span, timestamp *tout)	Adds an interval variable to a timestamp variable.	This function accepts a pointer **tin** pointing to the timestamp variable and a pointer **span** pointing to the interval variable, adds the interval to the timestamp, and saves the result timestamp in the variable that **tout** points to. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_sub_interval(timestamp *tin, interval *span, timestamp *tout)	Subtracts an interval variable from a timestamp variable.	This function subtracts the interval variable that **span** points to from the timestamp variable that **tin** points to, and then saves the result in the variable that **tout** points to. The function returns **0** on success and a negative value in case of error.

Examples

For details, see **Typical Application Development Examples**.

5.9 Compatibility Reference

Go Driver Compatibility

- The Go driver of the database does not adapt to mature ORM frameworks (such as XORM) in the industry.
- During the creation of a database connection, **opengauss**, **postgres**, or **postgresql** may be passed as the driver name.
- After the environment variable *GOENV_REGIST_ALL* is set to **false**, the Go driver of GaussDB and the Go driver of PG can coexist. If the environment variable *GOENV_REGIST_ALL* is set to **true** or the default value is used, the Go driver of GaussDB cannot coexist with the Go driver of PG. As a result, an error is reported for the code that depends on the Go driver of PG.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

ECPG Compatibility

ECPG provides the URL connection syntax which supports GaussDB and is PG-compatible.

Connection syntax:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

To support PG compatibility, **target** is set as follows:

- tcp:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- unix:postgresql://*hostname*[:*port*][/*dbname*][?*options*]

5.10 Appendix

5.10.1 JDBC

5.10.1.1 Data Type Mapping

Table 5-136 describes the mapping between data types, Java variable types, and JDBC type indexes (ORA: Oracle compatibility; MYSQL: MySQL compatibility).

- The non-multi-primary mode supports Oracle and MySQL compatibility.

Table 5-136 Mappings for JDBC data types

Compatibility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
ORA/MYSQL	oid	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
ORA/MYSQL	tinyint	java.lang.Integer	java.sql.Types.TINYINT
ORA/MYSQL	smallint	java.lang.Integer	java.sql.Types.SMALLINT
ORA/MYSQL	bigint	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	float4	java.lang.Float	java.sql.Types.REAL
ORA/MYSQL	float8	java.lang.Double	java.sql.Types.DOUBLE
ORA/MYSQL	char	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	bpchar	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character varying	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	varchar	java.lang.String	java.sql.Types.VARCHAR

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Compatibility Mode	GaussDB Data Type	Java Variable Type	JDBC Type Index
ORA/MYSQL	text	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	name	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	bytea	byte[]	java.sql.Types.BINARY
ORA/MYSQL	blob	java.sql.Blob	java.sql.Types.BLOB
ORA/MYSQL	clob	java.sql.Clob	java.sql.Types.CLOB
ORA/MYSQL	Boolean	java.lang.Boolean	java.sql.Types.BIT
ORA	Collection type	java.sql.Array	java.sql.Types.ARRAY
ORA	Array type	java.sql.Array	java.sql.Types.ARRAY
ORA	Record type	java.sql.Struct	java.sql.Types.STRUCT
MYSQL	date	java.sql.Date	java.sql.Types.DATE
ORA/MYSQL	time	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timetz	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timestamp	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	smalldatetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	timestamptz	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	refcursor	java.sql.ResultSet	java.sql.Types.REF_CURSOR java.sql.Types.OTHER

5.10.1.2 Data Type Conversion Rules

Table 5-136 describes the mapping between data types, Java variable types, and JDBC type indexes.

- The non-multi-primary mode supports ORA and MySQL compatibility.

NOTE

- The types sent to the server may differ from the types ultimately stored in database tables. To determine the stored types corresponding to the sent types listed below, see "SQL Reference > Type Conversion > Value Storage" in *Reference*. The type conversion rules also apply to MySQL compatibility.
- For details about JDBC connection parameters, see **Connection Parameter Reference**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-137 Data conversion rules of the set API in the PrepareStatement class

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
setNull(int parameterIndex, int sqlType)	ORA/MYSQL	int sqlType	java.lang.Integer	-	Depending on the value of the API parameter **sqlType** (see **Table 5-138**).	**sqlType** refers to the SQL type value defined in java.sql.Types.
void setNull(int parameterIndex, int sqlType, String typeName)	ORA/MYSQL	int sqlType	java.lang.Integer	Parameter **String typeName=null**	Refer to **Table 5-138**.	-
void setBoolean(int parameterIndex, boolean x)	ORA/MYSQL	boolean x	java.lang.Boolean	Connection parameter **use_boolean=true**	BOOL	-
	ORA/MYSQL	boolean x	java.lang.Boolean	Connection parameter **use_boolean=false** (default)	INT2	-
void setByte(int parameterIndex, byte x)	ORA/MYSQL	byte x	byte	-	INT2	-
void setShort(int parameterIndex, short x)	ORA/MYSQL	short x	java.lang.Short	-	INT2	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setInt(int parameterIndex, int x)	ORA/MYSQL	int x	java.lang.Integer	-	INT4	-
void setLong(int parameterIndex, long x)	ORA/MYSQL	long x	java.lang.Long	-	INT8	-
void setFloat(int parameterIndex, float x)	ORA	float x	java.lang.Float	Connection parameter **binaryTransfer=true**	FLOAT4	-
	ORA	float x	java.lang.Float	Connection parameter **binaryTransfer=false** (default) Connection parameter **setFloat=true**	FLOAT4	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	float x	java.lang.Float	Connection parameter **binaryTransfer=false** (default) Connection parameter **setFloat=false** (default)	FLOAT8	-
	MYSQL	float x	java.lang.Float	-	FLOAT4	-
void setDouble(int parameterIndex, double x)	ORA/MYSQL	double x	java.lang.Double	-	FLOAT8	-
void setBigDecimal(int parameterIndex, BigDecimal x)	ORA/MYSQL	BigDecimal x	java.math.BigDecimal	-	NUMERIC	-
void setString(int parameterIndex, String x)	ORA/MYSQL	String x	java.lang.String	Connection parameter **stringtype=varchar** (default)	VARCHAR	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	String x	java.lang.String	Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
void setBytes(int parameterIndex, byte x[])	ORA/MYSQL	byte x[]	[]byte	-	BYTEA	-
void setDate(int parameterIndex, java.sql.Date x, Calendar cal)	ORA/MYSQL	-	-	-	-	Same as **void setDate(int parameterIndex, java.sql.Date x)**.
void setDate(int parameterIndex, java.sql.Date x)	ORA	java.sql.Date x	java.lang.Date	Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=on**	DATEA	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.sql.Date x	java.lang.Date	Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=off**	UNSPECIFIED	-
	ORA/MYSQL	java.sql.Date x	java.lang.Date	Connection parameter **binaryTransfer=false** (default) Connection parameter **oracleCompatible** not being **true** and with **setDateTime** excluded in ORA-compatible mode	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	java.sql.Date x	java.lang.Date	Connection parameter **oracleCompatible=true** or with **setDateTime** included Connection parameter **binaryTransfer=false** (default) Database GUC parameter **mapping_date_to_datea=on**	DATEA	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	java.sql.Date x	java.lang.Date	Connection parameter **oracleCompatible=true** or with **setDateTime** included Connection parameter **binaryTransfer=false** (default) Database GUC parameter **mapping_date_to_datea=off**	TIMESTAMP	-
void setTime(int parameterIndex, java.sql.Time x, Calendar cal)	ORA/MYSQL	-	-	-	-	Same as **void setTime(int parameterIndex, java.sql.Time x)**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setTime(int parameterIndex, java.sql.Time x)	ORA/MYSQL	java.sql.Time x	java.lang.Time	Connection parameter **oracleCompatible** not being **true** and with **setDateTime** excluded in ORA-compatible mode	UNSPECIFIED	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Time time1 = Time.valueOf("12:30:45"); ps.setTime(1, time1);
	ORA/MYSQL	java.sql.Time x	PGTime	No time zone set in the **java.sql.Time x** parameter	TIME	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Time time1 = Time.valueOf("12:30:45"); PGTime time2 = new PGTime(time1.getTime()); ps.setTime(1, time2);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.sql.Time x	PGTime	Non-null time zone set in the **java.sql.Time x** parameter	TIMETZ	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Time time1 = Time.valueOf("12:30:45"); Calendar calendar = Calendar.getInstance(); PGTime time2 = new PGTime(time1.getTime(), calendar); ps.setTime(1, time2);
	ORA/MYSQL	java.sql.Time x	java.lang.Time	Parameter **java.sql.Time x** set to null	UNSPECIFIED	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); ps.setTime(1, null);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	java.sql.Time x	java.lang.Time	Connection parameter **oracleCompatible=true** or with **setDateTime** included	TIMETZ	-
void setTimestamp(int parameterIndex, java.sql.Timestamp x, Calendar cal)	ORA/MYSQL	-	-	-	-	Same as **void setTimestamp(int parameterIndex, java.sql.Timestamp x)**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setTimestamp(int parameterIndex, java.sql.Timestamp x)	ORA/MYSQL	java.sql.Timestamp x	java.sql.Timestamp	Connection parameter **oracleCompatible** not being **true** and with **setDateTime** excluded (default) in ORA-compatible mode	UNSPECIFIED	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Timestamp timestamp1 = Timestamp.valueOf("2024-01-01 12:30:45.888"); ps.setTimestamp(1, timestamp1);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.sql.Timestamp x	PGTimestamp	No time zone set in the **java.sql.Timestamp x** parameter	TIMESTAMP	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Timestamp timestamp1 = Timestamp.valueOf("2024-01-01 12:30:45.888"); PGTimestamp timestamp2 = new PGTimestamp(timestamp1.getTime()); ps.setTimestamp(1, timestamp2);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.sql.Timestamp x	PGTimestamp	Non-null time zone set in the **java.sql.Timestamp x** parameter	TIMESTAMPTZ	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); Timestamp timestamp1 = Timestamp.valueOf("2024-01-01 12:30:45.888"); Calendar calendar = Calendar.getInstance(); PGTimestamp timestamp2 = new PGTimestamp(timestamp1.getTime(), calendar); ps.setTimestamp(1, timestamp2);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.sql.Timestamp x	java.sql.Timestamp	Parameter **java.sql.Timestamp x** set to null	UNSPECIFIED	Example code: PreparedStatement ps = conn.prepareStatement("insert into ptest_table values (?)"); ps.setTimestamp(1, null);
	ORA	java.sql.Timestamp x	java.sql.Timestamp	Connection parameter **oracleCompatible=true** or with **setDateTime** included	TIMESTAMPTZ	-
void setAsciiStream(int parameterIndex, java.io.InputStream x)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype** not being **unspecified**	VARCHAR	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** not being null	VARCHAR	-
void setAsciiStream(int parameterIndex, java.io.InputStream x, int length)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** not being null	VARCHAR	-
void setAsciiStream(int parameterIndex, java.io.InputStream x, long length)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype** not being **unspecified**	VARCHAR	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** not being null	VARCHAR	-
void setBinaryStream(int parameterIndex, java.io.InputStream x)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=on**	BLOB	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=off**	BYTEA	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setBinaryStream(int parameterIndex, java.io.InputStream x, int length)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=on**	BLOB	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=off**	BYTEA	-
void setBinaryStream(int parameterIndex, java.io.InputStream x,long length)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=true**	BLOB	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Connection parameter **blobMode=false** (default)	BYTEA	-
void setCharacterStream(int parameterIndex, java.io.Reader reader,long length)	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Connection parameter **stringtype=unspecified**	UNSPECIFIED	Same as the **void setCharacterStream(int parameterIndex, java.io.Reader reader,int length)** API.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setCharacterStream(int parameterIndex, java.io.Reader reader,int length)	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
	ORA	java.io.Reader reader	java.io.Reader	Parameter **javaio.Reader reader** not being null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-
	MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** set to null	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setCharacterStream(int parameterIndex, java.io.Reader reader)	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** set to null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** set to null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **preferQueryMode=simple** Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **preferQueryMode=simple** Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.Reader reader	java.io.Reader	Parameter **java.io.Reader reader** not being null Connection parameter **preferQueryMode** not being **simple** Connection parameter **stringtype** not being **unspecified** (default)	TEXT	-
void setUnicodeStream(int parameterIndex, java.io.InputStream x, int length)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** set to null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-
	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	Parameter **java.io.InputStream x** not being null	VARCHAR	-
void setBlob(int parameterIndex, Blob x)	ORA/MYSQL	Blob x	java.sql.BLOB	-	BLOB	-
void setBlob(int parameterIndex, InputStream inputStream)	ORA/MYSQL	java.io.InputStream x	java.io.InputStream	-	BLOB	-
void setBlob(int parameterIndex, InputStream inputStream, long length)	ORA/MYSQL	InputStream inputStream	java.io.InputStream	-	BLOB	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
void setClob(int parameterIndex, Clob x)	ORA/MYSQL	Clob x	java.sql.CLOB	Parameter **Clob x** set to null	CLOB	-
	ORA/MYSQL	Clob x	java.sql.CLOB	Parameter **Clob x** not being null Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
	ORA/MYSQL	Clob x	java.sql.CLOB	Parameter **Clob x** not being null Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-
void setClob(int parameterIndex, Reader reader)	ORA/MYSQL	Reader reader	java.io.Reader	Connection parameter **stringtype=unspecified**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	Reader reader	java.io.Reader	Connection parameter **stringtype** not being **unspecified**	VARCHAR	-
void setClob(int parameterIndex, Reader reader, long length)	ORA/MYSQL	Reader reader	java.io.Reader	Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
	ORA/MYSQL	Reader reader	java.io.Reader	Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR	-
void setArray(int parameterIndex, Array x)	ORA/MYSQL	Array x	java.sql.Array	Parameter **Array x** set to null	UNSPECIFIED	-
	ORA/MYSQL	Array x	java.sql.Array	Parameter **Array x** not being null	Determined by the parameters for creating PGArray or GaussArray.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	Array x	GaussArray	-	-	-
	ORA	Array x	GaussArray	Parameter **Array x** set to null	UNSPECIFIED	-
void setRowId(int parameterIndex, RowId x)	ORA/MYSQL	RowId x	java.sql.RowId	-	ROWID	-
void setNString(int parameterIndex, String value)	ORA/MYSQL	String x	java.lang.String	Connection parameter **stringtype=varchar** (default)	VARCHAR	-
	ORA/MYSQL	String x	java.lang.String	Connection parameter **stringtype=unspecified**	UNSPECIFIED	-
void setSQLXML(int parameterIndex, SQLXML xmlObject)	ORA/MYSQL	SQLXML xmlObject	java.sql.SQLXML	-	XML	-
void setObject(int parameterIndex, Object x, int targetSqlType)	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** set to null	Refer to **Table 5-138**.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.SQLXML**	Refer to the **void setSQLXML(int parameterIndex, SQLXML xmlObject)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.INTEGER**	Refer to the **void setInt(int parameterIndex, int x)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TINYINT**	INT1	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.SMALLINT**	INT2	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BIGINT**	INT8	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.REAL**	Refer to the **void setFloat(int parameterIndex, float x)** API.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.FLOAT** Connection parameter **setFloat=true**	FLOAT4	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.FLOAT** Connection parameter **setFloat=false**	FLOAT8	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DOUBLE**	FLOAT8	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DECIMAL**	NUMERIC	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.NUMERIC**	NUMERIC	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.CHAR**	BPCHAR	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.VARCHAR**	VARCHAR	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.LONGVARCHAR** Parameter **Object x** set to the InputStream type	TEXT	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.LONGVARCHAR** Parameter **Object x** not being the InputStream type	VARCHAR	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=on**	DATEA	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=off**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=false** (default)	UNSPECIFIED	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=false** (default) Connection parameter **oracleCompatible** not being **true** and with **setDateTime** excluded	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=false** (default) Connection parameter **oracleCompatible=true** or with **setDateTime** included Database GUC	DATEA	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
				parameter **mapping_date_to_date=on**		

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** not being the LocalDate type Connection parameter **binaryTransfer=false** (default) Connection parameter **oracleCompatible=true** or with **setDateTime** included Database GUC	TIMESTAMP	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
				parameter **mapping_date_to_datea=off**		
	ORA	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** set to the LocalDate type Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=on**	DATEA	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DATE** Parameter **Object x** set to the LocalDate type Connection parameter **binaryTransfer=true** Database GUC parameter **mapping_date_to_datea=on**	DATE	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIME** Parameter **Object x** not being the LocalTime type	Refer to the **void setTime(int parameterIndex, java.sql.Time x)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIME** Parameter **Object x** set to the LocalTime type	TIME	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIMESTAMP** Parameter **Object x** not being the LocalDateTime type	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x, Calendar cal)** API.	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIMESTAMP** Parameter **Object x** set to the LocalDateTime type	TIMESTAMP	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIMESTAMP_WITH_TIMEZONE** Parameter **Object x** set to the OffsetDateTime type	TIMESTAMPTZ	-

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.TIMESTAMP_WITH_TIMEZONE** Parameter **Object x** set to the PGTimestamp type	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x, Calendar cal)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BIT** Connection parameter **use_boolean=true**	BOOL	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BIT** Connection parameter **use_boolean=false** (default)	INT2	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BOOLEAN** Connection parameter **use_boolean=true**	BOOL	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BOOLEAN** Connection parameter **use_boolean=false** (default)	INT2	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BINARY**, **Types.VARBINARY**, or **Types.LONGVARBINARY**	Determined by the data type of the **Object x** parameter (see **Table 5-139**).	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BLOB** Parameter **Object x** set to the BLOB type	Refer to the **void setBlob(int parameterIndex, Blob x)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.BLOB** Parameter **Object x** set to the InputStream type	Refer to the **void setLong(int parameterIndex, long x)** API.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.CLOB** Parameter **Object x** set to the CLOB type	Refer to the **void setClob(int parameterIndex, Clob x)** API.	-
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.DISTINCT**	UNSPECIFIED	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
	ORA/MYSQL	int targetSqlType	java.lang.Integer	Parameter **Object x** not being null Parameter **int targetSqlType=Types.ROWID**	ROWID	-
void setObject(int parameterIndex, Object x)	ORA/MYSQL	-	-	-	-	Determined by the type of the **Object x** parameter (see **Table 5-139**).
default void setObject(int parameterIndex, Object x, SQLType targetSqlType,int scaleOrLength)	ORA/MYSQL	-	-	-	-	Determined by the type of the **SQLType targetSqlType** parameter (see **Table 5-140**).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

JDBC API	Compatibility Mode	Target Parameter Name	Java Variable Type of the Target Parameter	Conditions	Type Sent to Databases	Remarks
default void setObject(int parameterIndex, Object x, SQLType targetSqlType)	ORA/MYSQL	-	-	-	-	Determined by the type of the **SQLType targetSqlType** parameter (see **Table 5-140**).
void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	ORA/MYSQL	-	-	-	-	Determined by the type of the **int targetSqlType** parameter (see **Table 5-140**).

Table 5-138 Database types corresponding to the java type of the setNull API

Java Type ID	Compatibility Mode	Conditions	Type Sent to Databases
Types.VARCHAR	ORA/MYSQL	Connection parameter **stringtype=unspecified**	UNSPECIFIED
Types.VARCHAR	ORA/MYSQL	Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR
Types.LONGVARCHAR	ORA/MYSQL	Connection parameter **stringtype=unspecified**	UNSPECIFIED

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Java Type ID	Compatibility Mode	Conditions	Type Sent to Databases
Types.LONGVARCHAR	ORA/MYSQL	Connection parameter **stringtype** not being **unspecified** (default)	VARCHAR
Types.DATE	ORA	Connection parameter **oracleCompatible=true** or with **setDateTime** included Database GUC parameter **mapping_date_to_datea=on**	DATEA
Types.DATE	ORA	Connection parameter **oracleCompatible=true** or with **setDateTime** included Database GUC parameter **mapping_date_to_datea=off**	TIMESTAMP
Types.DATE	ORA/MYSQL	-	DATE
Types.SQLXML	ORA/MYSQL	-	XML
Types.INTEGER	ORA/MYSQL	-	INT4
Types.TINYINT	ORA/MYSQL	-	INT1
Types.SMALLINT	ORA/MYSQL	-	INT2
Types.BIGINT	ORA/MYSQL	-	INT8
Types.REAL	ORA/MYSQL	-	FLOAT4
Types.DOUBLE	ORA/MYSQL	-	FLOAT8
Types.FLOAT	ORA/MYSQL	-	FLOAT8
Types.DECIMAL	ORA/MYSQL	-	NUMERIC
Types.NUMERIC	ORA/MYSQL	-	NUMERIC

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Java Type ID	Compatibility Mode	Conditions	Type Sent to Databases
Types.CHAR	ORA/MYSQL	-	BPCHAR
Types.TIME	ORA/MYSQL	-	UNSPECIFIED
Types.TIMESTAMP	ORA/MYSQL	-	UNSPECIFIED
Types.TIME_WITH_TIMEZONE	ORA/MYSQL	-	UNSPECIFIED
Types.TIMESTAMP_WITH_TIMEZONE	ORA/MYSQL	-	UNSPECIFIED
Types.BOOLEAN	ORA/MYSQL	-	BOOL
Types.BIT	ORA/MYSQL	-	BOOL
Types.BINARY	ORA/MYSQL	-	BYTEA
Types.VARBINARY	ORA/MYSQL	-	BYTEA
Types.LONGVARBINARY	ORA/MYSQL	-	BYTEA
Types.BLOB	ORA/MYSQL	-	BLOB
Types.CLOB	ORA/MYSQL	-	CLOB
Types.ARRAY	ORA/MYSQL	-	UNSPECIFIED
Types.DISTINCT	ORA/MYSQL	-	UNSPECIFIED
Types.STRUCT	ORA/MYSQL	-	UNSPECIFIED
Types.NULL	ORA/MYSQL	-	UNSPECIFIED
Types.OTHER	ORA/MYSQL	-	UNSPECIFIED

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-139 Object x types available in the setObject API

Type	Compatibility Mode	Conditions	Type Sent to Databases
null	ORA/MYSQL	-	Refer to the **setNull(int parameterIndex, int sqlType)** API in **Table 5-137**.
UUID	ORA/MYSQL	-	UUID
SQLXML	ORA/MYSQL	-	Refer to the **void setSQLXML(int parameterIndex, SQLXML xmlObject)** API in **Table 5-137**.
String	ORA/MYSQL	-	Refer to the **void setString(int parameterIndex, String x)** API in **Table 5-137**.
Character	ORA/MYSQL	-	Refer to the **void setString(int parameterIndex, String x)** API in **Table 5-137**.
RowId	ORA/MYSQL	-	Refer to the **void setRowId(int parameterIndex, RowId x)** API in **Table 5-137**.
BigDecimal	ORA/MYSQL	-	Refer to the **void setBigDecimal(int parameterIndex, BigDecimal x)** API in **Table 5-137**.
Short	ORA/MYSQL	-	Refer to the **void setShort(int parameterIndex, short x)** API in **Table 5-137**.
Integer	ORA/MYSQL	-	Refer to the **void setInt(int parameterIndex, int x)** API in **Table 5-137**.
Long	ORA/MYSQL	-	Refer to the **void setLong(int parameterIndex, long x)** API in **Table 5-137**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Compatibility Mode	Conditions	Type Sent to Databases
Float	ORA/MYSQL	-	Refer to the **void setFloat(int parameterIndex, float x)** API in **Table 5-137**.
Double	ORA/MYSQL	-	Refer to the **void setDouble(int parameterIndex, double x)** API in **Table 5-137**.
byte[]	ORA/MYSQL	-	Refer to the **void setBytes(int parameterIndex, byte x[])** API in **Table 5-137**.
Byte	ORA/MYSQL	-	Refer to the **void setByte(int parameterIndex, byte x)** API in **Table 5-137**.
java.sql.Date	ORA/MYSQL	-	Refer to the **void setDate(int parameterIndex, java.sql.Date x)** API in **Table 5-137**.
Time	ORA/MYSQL	-	Refer to the **void setTime(int parameterIndex, java.sql.Time x)** API in **Table 5-137**.
Timestamp	ORA/MYSQL	-	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x)** API in **Table 5-137**.
LocalDate	MYSQL	-	DATE
	ORA	Connection parameter **mapping_date_to_datea=on**	DATEA
	ORA	Connection parameter **mapping_date_to_datea=off**	TIMESTAMP
LocalTime	ORA/MYSQL	-	TIME

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Compatibility Mode	Conditions	Type Sent to Databases
LocalDateTime	ORA/MYSQL	-	TIMESTAMP
OffsetDateTime	ORA/MYSQL	-	TIMESTAMPTZ
java.util.Date	ORA/MYSQL	-	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x)** API in **Table 5-137**.
Boolean	ORA/MYSQL	-	Refer to the **void setBoolean(int parameterIndex, boolean x)** API in **Table 5-137**.
Blob	ORA/MYSQL	-	Refer to the **void setBlob(int parameterIndex, Blob x)** API in **Table 5-137**.
Clob	ORA/MYSQL	-	Refer to the **void setClob(int parameterIndex, Clob x)** API in **Table 5-137**.
Number	ORA/MYSQL	-	NUMERIC

Table 5-140 Execution APIs corresponding to the targetSqlType parameter of the setObject API

Value of targetSqlType	Compatibility Mode	Conditions	Type Sent to Databases
Types.SQLXML	ORA/MYSQL	-	Refer to the **void setSQLXML(int parameterIndex, SQLXML xmlObject)** API in **Table 5-137**.
Types.INTEGER	ORA/MYSQL	-	Refer to the **void setInt(int parameterIndex, int x)** API in **Table 5-137**.
Types.TINYINT	ORA/MYSQL	-	INT1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value of targetSqlType	Compatibility Mode	Conditions	Type Sent to Databases
Types.SMALLINT	ORA/MYSQL	-	Refer to the **void setShort(int parameterIndex, short x)** API in **Table 5-137**.
Types.BIGINT	ORA/MYSQL	-	Refer to the **void setLong(int parameterIndex, long x)** API in **Table 5-137**.
Types.REAL	ORA/MYSQL	-	Refer to the **void setFloat(int parameterIndex, float x)** API in **Table 5-137**.
Types.FLOAT	ORA/MYSQL	Connection parameter **setFloat=true**	Refer to the **void setFloat(int parameterIndex, float x)** API in **Table 5-137**.
	MYSQL	Connection parameter **setFloat=false** (default) Connection parameter **mCompatible** not including **pstatement**	Refer to the **void setFloat(int parameterIndex, float x)** API in **Table 5-137**.
	ORA	Connection parameter **setFloat=false** (default)	Refer to the **void setDouble(int parameterIndex, double x)** API in **Table 5-137**.
	MYSQL	Connection parameter **setFloat=false** (default) Connection parameter **mCompatible=true** or with **pstatement** included	Refer to the **void setDouble(int parameterIndex, double x)** API in **Table 5-137**.
Types.DOUBLE	ORA/MYSQL	-	Refer to the **void setDouble(int parameterIndex, double x)** API in **Table 5-137**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value of targetSqlType	Compatibility Mode	Conditions	Type Sent to Databases
Types.DECIMAL	ORA/MYSQL	-	Refer to the **void setBigDecimal(int parameterIndex, BigDecimal x)** API in **Table 5-137**.
Types.NUMERIC	ORA/MYSQL	-	Refer to the **void setBigDecimal(int parameterIndex, BigDecimal x)** API in **Table 5-137**.
Types.CHAR	ORA/MYSQL	-	BPCHAR
Types.VARCHAR	ORA/MYSQL	-	VARCHAR
Types.LONGVARCHAR	ORA/MYSQL	Parameter **Object x** set to the InputStream type	TEXT
	ORA/MYSQL	Parameter **Object x** not being the InputStream type	VARCHAR
Types.DATE	ORA/MYSQL	-	Refer to the **void setDate(int parameterIndex, java.sql.Date x)** API in **Table 5-137**.
Types.TIME	ORA/MYSQL	-	Refer to the **void setTime(int parameterIndex, java.sql.Time x)** API in **Table 5-137**.
Types.TIMESTAMP	ORA/MYSQL	-	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x)** API in **Table 5-137**.
Types.TIMESTAMP_WITH_TIMEZONE	ORA/MYSQL	Parameter **Object x** set to the OffsetDateTime or PGTimestamp type	Refer to the **void setTimestamp(int parameterIndex, java.sql.Timestamp x)** API in **Table 5-137**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value of targetSqlType	Compatibility Mode	Conditions	Type Sent to Databases
Types.BIT	ORA/MYSQL	Connection parameter **mCompatible** not being **true** and with **pstatement** excluded	Refer to the **void setBoolean(int parameterIndex, boolean x)** API in **Table 5-137**.
	MYSQL	Connection parameter **mCompatible=true** or with **pstatement** included	Refer to the **void setInt(int parameterIndex, int x)** API in **Table 5-137**.
Types.BOOLEAN	ORA/MYSQL	-	Refer to the **void setBoolean(int parameterIndex, boolean x)** API in **Table 5-137**.
Types.BINARY	ORA/MYSQL	-	Determined by the data type of the **Object x** parameter (see **Table 5-139**).
Types.VARBINARY	ORA/MYSQL	-	Determined by the data type of the **Object x** parameter (see **Table 5-139**).
Types.LONGVARBINARY	ORA/MYSQL	-	Determined by the data type of the **Object x** parameter (see **Table 5-139**).
Types.BLOB	ORA/MYSQL	Parameter **Object x** set to the BLOB type	Refer to the **void setBlob(int parameterIndex, Blob x)** API in **Table 5-137**.
	ORA/MYSQL	Parameter **Object x** set to the InputStream type	Refer to the **void setLong(int parameterIndex, long x)** API in **Table 5-137**.
Types.CLOB	ORA/MYSQL	Parameter **Object x** set to the CLOB type	Refer to the **void setClob(int parameterIndex, Clob x)** API in **Table 5-137**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value of targetSqlType	Compatibility Mode	Conditions	Type Sent to Databases
Types.ARRAY	ORA/MYSQL	Parameter **Object x** set to the array type	Refer to the **void setArray(int parameterIndex, Array x)** API in **Table 5-137**.
Types.DISTINCT	ORA/MYSQL	-	UNSPECIFIED
Types.ROWID	ORA/MYSQL	Parameter **Object x** set to the RowId type	Refer to the **void setRowId(int parameterIndex, RowId x)** API in **Table 5-137**.
	ORA/MYSQL	Parameter **Object x** set to the string type	ROWID
Types.OTHER	ORA/MYSQL	Parameter **Object x** set to the UUID type Server parameter **SERVER_VERSION > 8.3.0**	UUID
	ORA/MYSQL	Parameter **Object x** not being the PGobject or MAP type	UNSPECIFIED

Table 5-141 get APIs of the ResultSet class

Method	Processing Logic	Remarks
getString(int columnIndex)	These methods obtain and convert the corresponding JDBC type based on column OIDs, database types, and connection parameters. If conversion to a string type is required, they parse and process special data formats—such as hexadecimal data—from specific databases.	-
getString(String columnLabel)		-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getBoolean(int columnIndex)	For BOOL-type column OIDs with a data length of 1 and an ASCII value of 116 (lowercase letter 't'), **true** is returned; otherwise, **false** is returned. In other scenarios, byte data is first converted into a character string based on column OIDs and connection parameters. The resulting string is then parsed to determine whether it conforms to Boolean semantics (such as **"true"**, **"t"**, **"0"**, or **"1"**), and the appropriate result is returned.	-
getBoolean(String columnLabel)		-
getByte(int columnIndex)	These methods obtain and convert the corresponding JDBC type based on column OIDs and connection parameters. Based on this type, data is first converted to a string or long type and then forcibly converted to a byte type for return. For high-precision numeric objects, their decimal parts are truncated.	-
getByte(String columnLabel)		-
getShort(int columnIndex)	Based on column OIDs and connection parameters, data is first converted to a string type—with special characters such as $ ( ) removed—or a long type, and then forcibly converted to a short type for return. For high-precision numeric objects, their decimal parts are truncated.	-
getShort(String columnLabel)		-
getInt(int columnIndex)	For INT4-type column OIDs, data is directly converted to an int type for return. In other scenarios, the data is first converted to a string type—with special characters removed—or a long type, and then forcibly converted to an int type for return. For high-precision numeric objects, their decimal parts are truncated.	-
getInt(String columnLabel)		-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getLong(int columnIndex)	For INT8-type column OIDs, data is directly converted to a long type for return. In other scenarios, the data is first converted to a string type—with special characters removed—and then parsed and converted to a long type for return. For high-precision numeric objects, their decimal parts are truncated.	-
getLong(String columnLabel)		-
getFloat(int columnIndex)	For FLOAT4-type column OIDs, data is directly converted to a float type for return. In other scenarios, the data is first converted to a string type—with special characters removed—or a double type, and then forcibly converted to a float type for return.	-
getFloat(String columnLabel)		-
getDouble(int columnIndex)	For FLOAT8-type column OIDs, data is directly converted to a double type for return. In other scenarios, the data is first converted to a string type—with special characters removed—and then parsed and converted to a double type for return.	-
getDouble(String columnLabel)		-
getBigDecimal(int columnIndex, int scale)	These methods obtain and convert the corresponding JDBC type based on column OIDs and connection parameters. Based on this type, data is first converted to a string, number, or long type. Then a BigDecimal object is constructed and returned.	The four APIs call the same underlying function getBigDecimal(int columnIndex, int scale), where: When the caller is getBigDecimal(int columnIndex) or getBigDecimal(String columnLabel), **scale** is **-1**.
getBigDecimal(String columnLabel, int scale)		
getBigDecimal(int columnIndex)		
getBigDecimal(String columnLabel)		

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getBytes(int columnIndex)	Based on column OIDs and connection parameters, the byte array is directly retained and returned in certain scenarios. In other scenarios, data is first converted into a string corresponding to the JDBC type and then extracted as a byte array. Finally, the array is truncated based on the **Statement.setMaxFieldSize** limit before being returned.	-
getBytes(String columnLabel)		-
getDate(int columnIndex)	Based on column OIDs and connection parameters, data is directly converted to a timestamp or number of days in certain scenarios, and then converted to a date based on the applicable time zone before being returned. In other scenarios, the data is first converted into a string, and then parsed and constructed into a date object based on the time zone before being returned.	The four APIs call the same underlying function getDate(int columnIndex, java.util.Calendar cal), where: When the caller is getDate(int columnIndex) or getDate(String columnLabel), **cal** is null.
getDate(String columnLabel)		
getDate(int columnIndex, Calendar cal)		
getDate(String columnLabel, Calendar cal)		
getTime(int columnIndex)	Based on column OIDs and connection parameters, data is directly converted to the corresponding underlying data in certain scenarios, after which a time value is derived based on the applicable time zone. In other scenarios, the data is first converted into a string, and then parsed and constructed into a time object based on the time zone before being returned.	The four APIs call the same underlying function getTime(int columnIndex, java.util.Calendar cal), where: When the caller is getTime(int columnIndex) or getTime(String columnLabel), **cal** is null.
getTime(String columnLabel)		
getTime(int columnIndex, Calendar cal)		
getTime(String columnLabel, Calendar cal)		

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getTimestamp(int columnIndex)	Based on column OIDs and connection parameters, data is directly converted to the corresponding underlying data in certain scenarios, after which a timestamp value is derived based on the applicable time zone. In other scenarios, the data is first converted into a string, and then parsed and constructed into a timestamp object based on the time zone before being returned.	The four APIs call the same underlying function getTimestamp(int columnIndex, java.util.Calendar cal), where: When the caller is getTimestamp(int columnIndex) or getTimestamp(String columnLabel), **cal** is null.
getTimestamp(String columnLabel)		
getTimestamp(int columnIndex, Calendar cal)		
getTimestamp(String columnLabel, Calendar cal)		
getAsciiStream(int columnIndex)	Based on column OIDs and connection parameters, data is converted into a string, encoded via ASCII into a byte array, and encapsulated into a ByteArrayInputStream object for return.	-
getAsciiStream(String columnLabel)		-
getBinaryStream(int columnIndex)	Similar to getBytes, these methods obtain a byte array compliant with the **Statement.setMaxFieldSize** limit and encapsulate it into a ByteArrayInputStream object for return.	-
getBinaryStream(String columnLabel)		-
getObject(int columnIndex)	These methods obtain and convert the corresponding JDBC type based on column OIDs and connection parameters. If conversion to a string type is required, they apply appropriate processing to special data formats.	-
getObject(String columnLabel)		-
getCharacterStream(int columnIndex)	Internally, these methods call getString to obtain a string. Then they convert the string into a character array using toCharArray(), construct a CharArrayReader from this array, and return the Reader.	-
getCharacterStreamt(String columnLabel)		-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getRow()	This method obtains the row number that the cursor is pointing to in the current result set. The numbering begins at 1 for the first row, 2 for the second row, and so on.	-
getObject(int columnIndex, java.util.Map<String,Class<?>> map)	The JDBC driver currently does not support user-defined type mappings. When the input **map** is empty, it defaults to calling the getObject(int columnIndex) API as a fallback to return data.	-
getObject(String columnLabel, java.util.Map<String,Class<?>> map)		-
getRef(int columnIndex)	Not supported currently.	-
getRef(String columnLabel)		-
getBlob(int columnIndex)	These methods obtain the processed byte array based on column OIDs and connection parameters. Using this underlying data, they construct and return an instance object implemented based on the java.sql.Blob API.	-
getBlob(String columnLabel)		-
getClob(int columnIndex)	These methods obtain the processed byte array based on column OIDs and connection parameters. Using this underlying data, they construct and return an instance object implemented based on the java.sql.Clob API.	-
getClob(String columnLabel)		-
getArray(int columnIndex)	These methods obtain the processed byte array based on column OIDs and connection parameters. Using this underlying data, they construct and return an instance object implemented based on the java.sql.Array API.	-
getArray(String columnLabel)		-
getURL(int columnIndex)	Internally, these methods call getString to obtain a string. Using this string, they construct and return a java.net.URL object.	-
getURL(String columnLabel)		-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getRowId(int columnIndex)	Internally, these methods call getBytes to obtain a byte array. Using this underlying data, they construct and return an instance object implemented based on the java.sql.RowId API.	-
getRowId(String columnLabel)		-
getHoldability()	Not supported currently.	-
getNClob(int columnIndex)	Not supported currently.	-
getNClob(String columnLabel)		-
getSQLXML(int columnIndex)	Internally, these methods call getString to obtain a string. Using this string, they construct and return an instance object implemented based on the java.sql.SQLXML API.	-
getSQLXML(String columnLabel)		-
getNString(int columnIndex)	These methods follow the same processing logic as getString.	-
getNString(String columnLabel)		-
getNCharacterStream(int columnIndex)	Not supported currently.	-
getNCharacterStream(String columnLabel)		-
getObject(int columnIndex, Class<T> type)	Based on the input **type** parameter (Class type), these methods determine the conversion method for the byte data retrieved from the database and return the data in the specified type. (Certain types involve special processing logic.)	-
getObject(String columnLabel, Class<T> type)		-
getCursorName()	Null is returned.	-
getMetaData()	This method returns metadata descriptions for all columns in the current result set.	-
getFetchDirection()	This method returns the fetch direction of the current result set.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Processing Logic	Remarks
getFetchSize()	This method returns the fetch size of the current result set, that is, the number of rows that the JDBC driver obtains from the database in batches each time.	-
getType()	This method returns the type (cursor type) of the current result set.	-
getConcurrency()	This method returns the concurrency mode of the current result set.	-
getStatement()	This method returns the Statement object of the current result set.	-

NOTE

When GaussDB JDBC calls get APIs of the ResultSet class, the data decoding and type mapping logic may differ from Oracle Database in specific scenarios.

Switching between these underlying drivers can lead to differences in data representation at the Java application level, primarily in the following aspects:

- Type mapping policies: Different drivers interpret native database types (such as the generic NUMBER) differently. Some drivers enforce a strongly typed, strict mapping (for example, consistently to BigDecimal), whereas customized drivers—potentially for compatibility with legacy systems—may employ a more generalized mapping to Double or String.
- Precision and rounding boundaries: When processing high-precision decimals or extremely large numbers, drivers vary in their tolerance levels. Some drivers may silently truncate data or pad with zeros upon precision overflow, whereas more rigorous ones may throw boundary exceptions.
- Environment and context dependencies: When parsing pure numbers in sensitive data such as timestamps, different drivers may apply slightly varying algorithms and benchmarks to compensate for the local server timezone offset.
- Fallback policy for null values and dirty data: When the underlying database contains NULL or non-standard data formats, drivers differ significantly in exception blocking mechanisms and default value padding (fallback logic).
- Tolerance and rules for implicit type conversion: When the get APIs of the ResultSet class do not strictly match the underlying column types of the database—for example, getString() is called for a numeric/date column, or getInt() is called for a string column—drivers vary in implicit conversion mechanisms. Some drivers demonstrate high tolerance by performing automatic cross-type forced conversion at the bottom layer, whereas stricter ones may throw type mismatch exceptions or use fundamentally distinct default serialization formats (for example, distinct default output styles for time) during implicit conversion to strings.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.10.1.3 Log Management

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

- Use the SLF4J log framework for interconnecting with applications.
- Use the JdkLogger log framework for interconnecting with applications.

SLF4J and JdkLogger are mainstream frameworks for Java application log management in the industry. For details about how to use these frameworks, see the official documents (SLF4J: **https://www.slf4j.org/manual.html**; JdkLogger: **https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html**).

Method 1: Use the SLF4J log framework for interconnecting with applications.

When a connection is set up, **logger=Slf4JLogger** needs to be configured in the URL. This mode supports log management and control. The SLF4J can implement powerful log management and control functions through related configurations in files. This method is recommended.

The SLF4J may be implemented by using Log4j or Log4j2. When Log4j is used to implement SLF4J, you need to add the **log4j-*.jar**, **slf4j-api-*.jar**, and **slf4j-log4*-*.jar** packages (*varies according to versions). In addition, specify the path (absolute or relative path) of the **log4j.properties** configuration file.

If the Log4j2 is used to implement the SLF4J, you need to add the following JAR packages: **log4j-api-*.jar**, **log4j-core-*.jar**, **log4j-slf4j18-impl-*.jar**, and **slf4j-api-*-alpha1.jar** (*varies according to versions). In addition, specify the path (absolute or relative path) of the **log4j2.xml** configuration file.

NOTICE

This method depends on the general SLF4J APIs, such as **org.slf4j.LoggerFactory.getLogger(String name)**, **org.slf4j.Logger.debug(String var1)**, **org.slf4j.Logger.info(String var1)**, **org.slf4j.Logger.warn(String warn)**, and **org.slf4j.Logger.warn(String warn)**. If these APIs are changed, logs cannot be recorded.

Example in non-multi-primary mode:

```
// Specify the path of the configuration file log4j.properties or log4j2.xml.

// Configure log recorder properties—such as log level and output target—in the log4j.properties file. These properties will be loaded into applications when you call the configure method of PropertyConfigurator and pass the path of the log4j.properties file.
//PropertyConfigurator.configure("log4j.properties");

// Create the Log4j2 configuration source, specify the path of the log4j2.xml configuration file, and initialize the log system.
//ConfigurationSource source = new ConfigurationSource(new FileInputStream("log4j2.xml"));
//Configurator.initialize(null, source);

public static Connection GetConnection(String username, String passwd){
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    String sourceURL = "jdbc:gaussdb://$ip.$port/database?logger=Slf4JLogger";
    Connection conn = null;

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

log4j.properties configuration file:

```
log4j.logger.com.huawei.gaussdb.jdbc=ALL, log_gsjdbc

# Default file output configuration
log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true
log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE
log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding = UTF-8
```

log4j2.xml configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
    <appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d %p %t %c - %m%n"/>
        </Console>
        <File name="FileTest" fileName="test.log">
            <PatternLayout pattern="%d %p %t %c - %m%n"/>
        </File>
        <!-- JDBC driver log file output configuration. Log rewinding is supported. When the log size exceeds
10 MB, a new file is created. The new file is named in the format of yyyy-mm-dd-file ID. -->
        <RollingFile name="RollingFileJdbc" fileName="gsjdbc.log" filePattern="%d{yyyy-MM-dd}-%i.log">
            <PatternLayout pattern="%d %p %t %c - %m%n"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="10 MB"/>
            </Policies>
        </RollingFile>
    </appenders>
    <loggers>
        <root level="all">
            <appender-ref ref="Console"/>
            <appender-ref ref="FileTest"/>
        </root>
        <!-- JDBC driver logs. The log level is all. All logs can be viewed and exported to the gsjdbc.log file. -->
        <logger name="com.huawei.gaussdb.jdbc" level="all" additivity="false">
            <appender-ref ref="RollingFileJdbc"/>
        </logger>
    </loggers>
</configuration>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- In the configuration file, you can specify the configuration paths for JDBC and upper-layer services to store JDBC logs and service logs to different files.
- When multiple GaussDB data sources are configured in the same project, if **logger** is set to **Slf4JLogger** in some services, it will affect the logging configuration of each other. In this case, you are advised to configure the connection strings in a unified manner.

Method 2: Use the JdkLogger log framework for interconnecting with applications.

You can use the **logging.properties** configuration file or directly configure the parameters in the sample code.

- The default Java logging framework stores its configurations in a file named **logging.properties**. Java installs the global configuration file in the folder in the Java installation directory. The **logging.properties** file can also be created and stored with a single project.

  logging.properties configuration file:

  ```
  # Specify the processing program as a file.
  handlers= java.util.logging.FileHandler

  # Specify the default global log level.
  .level= ALL

  # Specify the log output control standard.
  java.util.logging.FileHandler.level=ALL
  java.util.logging.FileHandler.pattern = gsjdbc.log
  java.util.logging.FileHandler.limit = 500000
  java.util.logging.FileHandler.count = 30
  java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
  java.util.logging.FileHandler.append=false
  ```

- Configure the parameters in the sample code:

  ```
  System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
  FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
  Formatter formatter = new SimpleFormatter();
  fileHandler.setFormatter(formatter);
  Logger logger = Logger.getLogger("com.huawei.gaussdb.jdbc");
  logger.addHandler(fileHandler);
  logger.setLevel(Level.ALL);
  logger.setUseParentHandlers(false);
  ```

Link Trace Function

The GaussDB JDBC driver provides the application-to-database link trace function to associate discrete SQL statements on the database side with application requests. This function requires application developers to implement the com.huawei.gaussdb.jdbc.log.Tracer API class and specify the full class name of the API implementation class in the URL.

URL example:

```
String url = "jdbc:gaussdb://$ip:$port/database?traceInterfaceClass=xxx.xxx.xxx.gaussdbTraceImpl";
```

The com.huawei.gaussdb.jdbc.log.Tracer API class is defined as follows:

```
public interface Tracer {
// Retrieves the value of traceId.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
String getTraceId();
}
```

The following is an example of the com.huawei.gaussdb.jdbc.log.Tracer API implementation class:

```
import com.huawei.gaussdb.jdbc.log.Tracer;

public class gaussdbTraceImpl implements Tracer {
    private static MDC mdc = new MDC();

    private final String TRACE_ID_KEY = "traceId";

    public void set(String traceId) {
        mdc.put(TRACE_ID_KEY, traceId);
    }

    public void reset() {
        mdc.clear();
    }

    @Override
    public String getTraceId() {
        return mdc.get(TRACE_ID_KEY);
    }
}
```

The following is an example of context mapping which is used to store trace IDs generated for different requests:

```
import java.util.HashMap;

public class MDC {
    static final private ThreadLocal<HashMap<String, String>> threadLocal = new ThreadLocal<>();

    public void put(String key, String val) {
        if (key == null || val == null) {
            throw new IllegalArgumentException("key or val cannot be null");
        } else {
            if (threadLocal.get() == null) {
                threadLocal.set(new HashMap<>());
            }
            threadLocal.get().put(key, val);
        }
    }

    public String get(String key) {
        if (key == null) {
            throw new IllegalArgumentException("key cannot be null");
        } else if (threadLocal.get() == null) {
            return null;
        } else {
            return threadLocal.get().get(key);
        }
    }

    public void clear() {
        if (threadLocal.get() == null) {
            return;
        } else {
            threadLocal.get().clear();
        }
    }
}
```

Take the service **traceId** as an example. The prerequisite is that the table **test_trace_id (id int,name varchar(20))** is created.

```
String traceId = UUID.randomUUID().toString().replaceAll("-", "");
gaussdbTraceImpl gaussdbTrace = new gaussdbTraceImpl();
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdbTrace.set(traceId);
Connection con = DriverManager.getConnection(url, user, password);
pstm = con.prepareStatement("SELECT * FROM test_trace_id WHERE id = ?");
pstm.setInt(1, 1);
pstm.execute();
pstm = con.prepareStatement("INSERT INTO test_trace_id VALUES(?,?)");
pstm.setInt(1, 2);
pstm.setString(2, "test");
pstm.execute();
gaussdbTrace.reset();
```

NOTE

- When the link trace function is used, the link function at the application layer is guaranteed by services.
- The application must expose the API for obtaining **traceId** to the JDBC and configure the API implementation class to the JDBC connection string.
- SQL statements of the same request must use the same **traceId**.
- The value of **traceId** specified by the application cannot exceed 32 bytes. Otherwise, the excess bytes will be truncated.

5.10.1.4 Troubleshooting

5.10.1.4.1 Incorrect batchMode Settings

Symptom

In the **batchMode=on** and **reWriteBatchedInserts=true** setup, using JDBC to insert data in batches will trigger a message indicating that the number of bound parameters does not match the number of parameters required by the statement.

```
bind message supplies * parameters, but prepared statement "" requires *
```

Example:

```
// The connection object conn has been created using a URL that includes batchMode=on&reWriteBatchedInserts=true.
// Bind parameters in batches and then execute the statement. The number of bound parameters does not match the number of columns in the rewritten INSERT statement. As a result, an exception is reported.
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6

PreparedStatement stmt = conn.prepareStatement("INSERT INTO test_tbl VALUES (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.executeBatch();
```

Cause Analysis

When **reWriteBatchedInserts** is set to **true**, multiple SQL statements are combined into one. As a result, the number of reserved parameter columns in the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

statement changes. If **batchMode** is set to **on**, parameters are bound based on the SQL statements before combination. As a result, the number of bound parameters is inconsistent with the number of parameters required by the statement.

Solution

If **reWriteBatchedInserts** is set to **true**, set **batchMode** to **off**.

5.10.1.4.2 Error Is Reported When Verification Is Enabled for Data Insertion in the Hibernate Framework

Symptom

The customer migrates data from the ORA-compatible database to GaussDB. The table structure is migrated using the DRS tool. After the migration, the original service code of the customer is unavailable, and an error is reported when the Hibernate framework verifies the table structure.

```
Schema-validation: wrong column type encountered in column [execute_time] in table [abnormal_event]; found [int8 (Types#BIGINT)], but expecting [int4 (Types#INTEGER)]
```

Example:

- The structure of the **Student** table in the ORA-compatible database is as follows:
  ```
  id number(15,0)
  sid number(15,0)
  age number(10,0)
  ```
- After migrated to GaussDB using DRS, the **Student** table structure is as follows:
  ```
  id bigint
  sid bigint
  age Integer
  ```
- When the **Student** entity class information in the actual database is as follows:
  ```
  Long id
  Long sid
  Long age  -- An error is reported when the Hibernate framework verifies the table definition.
  ```

Configuration file **hibernate.cfg.xml**:

```
<hibernate-configuration>
   <session-factory>
      <!-- GaussDB connection information -->
      <property name="connection.driver_class">com.huawei.gaussdb.jdbc.Driver</property>
      <property name="connection.url">jdbc:gaussdb://x.x.x.x:xx/postgres?currentSchema=public</property>
      <property name="connection.username">xxxxxx</property>
      <property name="connection.password">xxxxxx</property>

      <!-- The following configurations are optional: -->

      <!-- Specify whether to support dialects. -->
      <!-- In PG-compatible mode, the following configurations must be in place: -->
       <property name="dialect">org.hibernate.dialect.PostgreSQL92Dialect</property>
      <!-- In ORA-compatible mode, the following configurations are recommended: -->
<!--       <property name="dialect">org.hibernate.dialect.OracleDialect</property>-->

      <property name="met"></property>

      <!-- Specify whether to print SQL statements when CURD is executed. -->
      <property name="show_sql">true</property>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        <!-- The table is created automatically. -->
<!--        <property name="hbm2ddl.auto">create</property>-->
        <!-- Enable the verification when inserting data. -->
        <property name="hbm2ddl.auto">validate</property>
        <!-- Disable the verification. -->
<!--        <property name="hbm2ddl.auto">none</property>-->

        <!-- Resource registration (entity class mapping file) -->
        <mapping resource="student.xml"/>
    </session-factory>
</hibernate-configuration>
```

Main function for inserting data:

```
// Create the object to be tested.
Student student = new Student();
student.setId(20L);
student.setAge(222L);
student.setSid(222L);

// Start a transaction and obtain an object based on the session.
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

// Save data using sessions.
session.save(student);

// Commit a transaction.
transaction.commit();

// After the operation is complete, close the session connection object.
session.close();
```

Cause Analysis

- Reason why data is successfully inserted into an ORA-compatible database and passes verification: When verifying data, the Hibernate framework compares **TypeCode** of the inserted data type java.lang.Long with **TypeCode** of the number type in the table definition. If they are inconsistent, Hibernate then compares **sqlType** of the inserted long type with **Type** of the table definition and converts the long type into **number(19,0)** (this mapping is maintained by org.hibernate.dialect.OracleDialect). Because the table definition uses the number type and **number(19,0)** also begins with **number**, the verification succeeds. When working with an ORA-compatible database, you can use the java.lang.Long type to insert any integer data involved in service code.
- Reason why the GaussDB verification is different but data is successfully inserted after users bypass the verification: When the Hibernate framework verifies the inserted data, the service code is of the java.lang.Long type, which matches **TypeCode** of the table data type bigint. Therefore, no error is reported. However, a mismatch occurs when the table data type is integer. In this case, Hibernate first compares **TypeCode** of the service code's java.lang.Long type with **TypeCode** of the table data type integer. If they differ, Hibernate converts the long type into int8 and the integer type into int4, following the mapping maintained by org.hibernate.dialect.PostgreSQLDialect. Since these types do not match, the verification fails. When working with GaussDB, you must use the Java integer type corresponding to your table type to insert any integer data involved in service code.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Solution

You can use the following methods to rectify the fault:

- Disable the verification function.
- Modify the service code at the customer side and use the corresponding Java integer for different table data types.

5.10.1.4.3 Error Is Reported or Connection Is Blocked in SSL Mode

Symptom

When JDBC establishes an SSL connection, strong random numbers are obtained on the client. During this process, the following issues may arise:

Scenario 1: If **loginTimeout** is specified in the connection string, the "Connection attempt timed out" error may be thrown.

Scenario 2: If **loginTimeout** is omitted in the connection string, the connection establishment may be blocked. The blocking stack, as observed by the JDK command **jstack** *<pid>*, is shown in the figure below.

```
"Thread-0" #18 prio=5 os_prio=0 tid=0x00007f2ad0385000 nid=0x5429 runnable [0x00007f2aa069b000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileInputStream.readBytes(Native Method)
        at java.io.FileInputStream.read(FileInputStream.java:255)
        at sun.security.provider.NativePRNG$RandomIO.readFully(NativePRNG.java:424)
        at sun.security.provider.NativePRNG$RandomIO.ensureBufferValid(NativePRNG.java:526)
        at sun.security.provider.NativePRNG$RandomIO.implNextBytes(NativePRNG.java:545)
        - locked <0x000000067273a950> (a java.lang.Object)
        at sun.security.provider.NativePRNG$RandomIO.access$400(NativePRNG.java:331)
        at sun.security.provider.NativePRNG$Blocking.engineNextBytes(NativePRNG.java:268)
        at java.security.SecureRandom.nextBytes(SecureRandom.java:468)
        at java.security.SecureRandom.next(SecureRandom.java:491)
        at java.util.Random.nextInt(Random.java:329)
        at sun.security.ssl.SSLContextImpl.engineInit(SSLContextImpl.java:106)
        at javax.net.ssl.SSLContext.init(SSLContext.java:282)
        at org.postgresql.ssl.LibPQFactory.<init>(LibPQFactory.java:175)
        at org.postgresql.core.SocketFactoryFactory.getSslSocketFactory(SocketFactoryFactory.java:62)
        at org.postgresql.ssl.MakeSSL.convert(MakeSSL.java:33)
        at org.postgresql.core.v3.ConnectionFactoryImpl.enableSSL(ConnectionFactoryImpl.java:723)
        at org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect(ConnectionFactoryImpl.java:203)
        at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:330)
        at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:58)
        at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:357)
```

Cause Analysis

The random number generation on the client is too slow to meet product requirements. The entropy source is insufficient. As a result, the service fails to be started. This problem exists in some Linux environments.

Solution

Solution 1: Start the haveged service on the client and increase the entropy value of the system entropy pool to improve the speed of reading random numbers. The startup command is as follows:

```
systemctl start haveged
```

Method 2: Adjust the JDK configuration on the client.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

NOTICE

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

5.10.2 ODBC

5.10.2.1 Configuration Parameters

This section explains parameters in the **odbc.ini** file. For details, see **Table 5-142**.

Table 5-142 Configuration parameters in the odbc.ini file

Parameter	Description	Example
[DSN]	Name of the data source.	[gaussdb]
Driver	Driver name, corresponding to **DriverName** in **odbcinst.ini**.	Driver=GaussMPP
Servername	• Server IP address. You can configure multiple IP addresses. Both IPv4 and IPv6 addresses are supported.	Servername=127.0.0.1
Database	Name of the database to connect to.	Database=db1
Username	Name of the database user.	Username=omm

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
Password	Password for the database user. **NOTE** The ODBC driver clears in-memory passwords to ensure that user credentials are not retained in memory post-connection. However, configuring this parameter may cause passwords to remain in memory for extended periods due to UnixODBC's caching of data source files. It is advisable to pass passwords to the relevant APIs during application connection rather than writing them in the data source configuration file. Additionally, the memory area holding the passwords should be promptly cleared after a successful connection. **NOTICE** When entering a password in the configuration file, adhere to the following HTTP rules: 1. Characters must comply with URL encoding specifications, such as writing the exclamation mark (!) as "%21" and the percent sign (%) as "%25". Special attention should be paid to these characters. 2. The plus sign (+) will be replaced by a space.	Password=**********
Port	Port number of the server. When load balancing is enabled, multiple port numbers can be configured, each corresponding to an IP address. If only one port number is configured in the presence of multiple IP addresses under load balancing, all the IP addresses will default to sharing the same port number—that is, the one specified.	Port=8000
Sslmode	Specifies whether to enable SSL. **NOTE** The descriptions of available SSL modes can be found in **Table 5-143**.	Sslmode=allow

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
Debug	Specifies whether to enable the debug mode and, if enabled, the log output level. Value range: 0 to *INT_MAX*. ● **0**: The debug mode is disabled. ● > 0: The debug mode is enabled. Mylogs will be output for the gsqlODBC driver and stored in the **/tmp/** directory with file names like **mylog_*xxx*.log**. ● **1**: Only FATAL logs are output. ● **2**: FATAL and ERROR logs are output. ● **3**: FATAL, ERROR, and WARN logs are output. ● **4**: FATAL, ERROR, WARN, and INFO logs are output. ● ≥ 5: Logs at all levels are output, including TRACE logs. The default value is **0**. **NOTE** 1. FATAL: reports fatal errors. 2. ERROR: reports exceptions. 3. WARN: reports warnings. 4. INFO: records key information during program runtime. 5. TRACE: provides debugging information for developers.	Debug=5

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
CommLog	Specifies whether to enable backend communication logging and, if enabled, the log output level. Value range: 0 to *INT_MAX*. ● **0**: Backend communication logging is disabled. ● > 0: Backend communication logging is enabled. Qlogs will be output for the gsqlODBC driver and stored in the **/tmp/** directory with file names like **gsqlodbc**_*xxx*.**log**. ● **1**: Only ERROR logs are output. ● **2**: ERROR and INFO logs are output. ● ≥ 3: ERROR, INFO, and TRACE logs are output. The default value is **0**. **NOTE** 1. ERROR: reports exceptions. 2. INFO: records key information during program runtime. 3. TRACE: provides debugging information for developers.	CommLog=1
UseServerSidePrepare	Specifies whether to enable the extended query protocol for the database. Value range: **0** and **1**. ● **0**: disabled. ● **1**: enabled. The default value is **1**.	UseServerSidePrepare=1
UseBatchProtocol	Specifies whether to enable the batch query protocol. (Enabling it helps enhance DML operation performance.) Value range: **0** and **1**. ● **0**: The batch query protocol is disabled (mainly used for communication with earlier database versions). ● **1**: The batch query protocol is enabled, provided that the GUC parameter **support_batch_bind** is set to **on**. The default value is **1**.	UseBatchProtocol=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
ForExtensionConnector	Specifies whether to send the savepoint. Pay attention to this setting when dealing with savepoint-related issues. Value range: **0** and **1**. • **0**: The savepoint is sent. • **1**: The savepoint is not sent. The default value is **1**.	ForExtensionConnector=1
ConnectionExtraInfo	Specifies whether to display the driver deployment path and process owner in the GUC parameter **connection_info**. Value range: **0** and **1**. • **0**: disabled. • **1**: enabled. The default value is **0**. **NOTE** When set to **1**, the ODBC driver reports the deployment path and process owner of the current driver to the database and records this information in the GUC parameter **connection_info**. You can retrieve this information by referring to "System Catalogs and System Views > System Views > Other System Views > PG_STAT_ACTIVITY" and "PGXC_STAT_ACTIVITY" in *Reference*.	ConnectionExtraInfo=1
BoolsAsChar	Specifies whether to process Boolean values as characters. Value range: **0** and **1**. • **0**: Boolean values are mapped to the SQL_BIT type. • **1**: Boolean values are mapped to the SQL_CHAR type. The default value is **1**.	BoolsAsChar = 1
RowVersioning	Specifies whether applications are allowed to check for prior modifications made by other users to the same row of data before updating it. Value range: **0** and **1**. • **0**: Applications are not allowed to perform this check. • **1**: Applications are allowed to perform this check. The default value is **0**.	RowVersioning=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
ShowSystemTables	Specifies whether to treat system catalogs as regular SQL tables by default. Value range: **0** and **1**. ● **0**: The driver does not treat system catalogs as regular SQL tables by default. ● **1**: The driver treats system catalogs as regular SQL tables by default. The default value is **0**.	ShowSystemTables=1
AutoBalance	Specifies whether to enable load balancing for ODBC connections. Value range: **0** and **1**. ● **0**: Load balancing is disabled. ● **1**: Load balancing is enabled. The default value is **0**. **NOTE** ● When the version of the connected database is earlier than 505.2.0.SPC0200, load balancing is not supported in standby clusters. ● Load balancing can be enabled only for a single DSN.	AutoBalance=1
RefreshCNListTime	Time for refreshing the CN list. This parameter can be configured when load balancing is enabled. The value is an integer, in seconds. Value range: 0 to *INT_MAX*. ● **0**: This parameter is disabled. ● > 0: The set value determines how often the CN list is refreshed. The default value is **10**.	RefreshCNListTime=5
Priority	This parameter can be configured when load balancing is enabled. Value range: **0** and **1**. ● **0**: This parameter is disabled. ● **1**: This parameter is enabled. The default value is **0**. **NOTE** When **Priority** is enabled, all application-initiated connections are preferentially routed to the CNs specified in the configuration file. If none of the configured CNs is available, the connections will be redirected to the remaining CNs.	Priority=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
UsingEip	This parameter can be configured when load balancing is enabled. Value range: **0** and **1**. • **0**: This parameter is disabled. • **1**: This parameter is enabled. The default value is **0**. **NOTE** This setting determines whether to use service IP addresses (corresponding to the **node_host1** and **node_port1** columns in the pgxc_node system catalog) for load balancing. When **UsingEip** is enabled, service IP addresses are used for load balancing. When it is disabled, data IP addresses (corresponding to the **node_host** and **node_port** columns in the pgxc_node system catalog) are used instead.	UsingEip=1
MaxCacheQueries	Number of prepared statements cached for each connection. Value range: 0–4096. The default value is **0**. **NOTE** If this parameter is set to **0**, the prepared statement cache pool is disabled on the client. If set to a value greater than 4096, the parameter is capped at **4096**. When the number of executed statements exceeds the limit specified by **MaxCacheQueries**, the least recently used statements are evicted from the cache.	MaxCacheQueries=128
MaxCacheSizeMiB	Total size of prepared statements cached for each connection. This parameter takes effect when **MaxCacheQueries** is greater than 0. Value range: 0–4096. The default value is **1**. **NOTE** When the total size of cached statements exceeds the limit specified by **MaxCacheSizeMiB**, the least recently used statements are evicted from the cache. If set to a value greater than 4096 (MB), the parameter is capped at **4096**.	MaxCacheSizeMiB=10

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
TcpUserTimeout	Maximum duration that transmitted data can remain unacknowledged before the TCP connection is forcibly closed. This parameter is intended for OSs that support the **TCP_USER_TIMEOUT** socket option. Value range: 0 to *INT_MAX*. The default value is **0**. **NOTE** When set to **0**, the system default will be applied. Ignore this parameter for UDS connections. Its unit is milliseconds.	TcpUserTimeout=5000
StandbyRead	Specifies whether to enable the read-on-standby mode in a distributed system. Value range: **0** and **1**. • **0**: The read-on-standby mode is disabled. • **1**: The read-on-standby mode is enabled. The default value is **0**.	StandbyRead=1
Pqopt	libpq connection parameters, separated by spaces. For details about libpq parameters, see **Connection Parameters**.	Pqopt=keepalives=0
KeepaliveTime	Duration of inactivity (in seconds) after which TCP will send a keepalive message to the server. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. When set to **0** or left unset, the system default will be applied. **NOTE** To disable the keepalive functionality, set **keepalives** to **0** in Pqopt parameters.	KeepaliveTime=2
KeepaliveInterval	Interval (in seconds) for retransmitting a TCP keepalive message that is not acknowledged by the server. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. When set to **0** or left unset, the system default will be applied.	KeepaliveInterval=2

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
KeepaliveCount	Number of times that a keepalive message can be retransmitted through TCP. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. When set to **0** or left unset, the system default will be applied.	KeepaliveCount=2
SocketTimeout	Socket read/write timeout period that counts after a connection is successfully established between the client and the server. Measured in seconds, it defaults to **0**. **NOTE** This parameter includes the timeout for database statement execution. Setting it too low may cause legitimate operations to time out prematurely. Therefore, the default is set to **0**, but it is advisable to configure a non-zero value according to specific scenarios.	SocketTimeout=5
SocketTimeoutInConnect	Socket read/write timeout period in the client-server connection establishment phase, following a successful three-way handshake by TCP. Measured in seconds, it defaults to **5**. **NOTE** Different from **SocketTimeout**, this parameter governs the post-handshake client-server connection establishment phase. Considering the potential for blocking during this phase, it is essential to configure an appropriate socket read/write timeout period that does not impact the timeout of statement execution.	SocketTimeoutInConnect=5
CancelTimeout	Timeout period for an application to send a cancel message. Measured in seconds, it defaults to **0**.	CancelTimeout=5
TcpSYNRetries	Maximum number of SYN retransmissions allowed during the TCP three-way handshake. If the connection still fails after reaching this limit, an error will be reported. The default value is **0**. **NOTE** Available on OSs that support the socket option **TCP_SYNCNT**, this parameter specifies the number of times that the client can retransmit failed SYN packets during the three-way handshake phase. When set to **0**, the system default will be applied. Ignore this parameter for UDS connections.	TcpSYNRetries=3

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
TextAsLongVarchar	Specifies whether to map the text type on the database kernel side to the SQL_LONGVARCHAR or SQL_VARCHAR type on the driver side. Value range: **0** and **1**. ● **0**: The text type on the kernel side is mapped to the SQL_VARCHAR type on the driver side. ● **1**: The text type on the kernel side is mapped to the SQL_LONGVARCHAR type on the driver side. The default value is **1**.	TextAsLongVarchar=1
MaxLongVarcharSize	Maximum length allowed for the SQL_LONGVARCHAR type on the driver side. Value range: 0 to *INT_MAX*. The default value is **8190**.	MaxLongVarcharSize=8190
MaxVarcharSize	Maximum length allowed for the SQL_VARCHAR type on the driver side. Value range: 0 to *INT_MAX*. The default value is **255**.	MaxVarcharSize=255
TargetServerType	Type of host to connect to. The connection succeeds only when the actual host type matches the configured one. The setting principles are as follows: ● **primary** (default value): The connection is limited to CNs in the cluster. ● **cluster-primary**: The connection is limited to CNs in the primary cluster. ● **cluster-mainnode**: The connection is limited to CNs in standby clusters. **NOTE** The **cluster-primary** and **cluster-mainnode** options are supported only in streaming DR scenarios.	TargetServerType=cluster-primary
EnableHostCache	Specifies whether to enable the node cache functionality. ● **0**: disabled. ● **1**: enabled. The default value is **1**. **NOTE** The node cache does not take effect when a single IP address is configured in the connection string.	EnableHostCache=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-143 SSL modes

SSL Mode	Enable SSL Encryption	Description
disable	No	SSL connections are not used. This is the default.
allow	Probably	SSL connections are used when required by the database server. However, this mode does not verify the authenticity of the database server.
prefer	Probably	SSL connections are preferred if the database supports SSL. However, this mode does not verify the authenticity of the database server.
require	Yes	SSL connections are required. This mode only encrypts data but does not verify the authenticity of the database server.
verify-ca	Yes	SSL connections are required. The server must present a certificate issued by a trusted CA. Currently, Windows-based ODBC does not support certificate authentication.
verify-full	Yes	SSL connections are required. In addition to the scope of **verify-ca**, this mode verifies that the database's host name matches the name in the certificate. GaussDB does not support this mode. Currently, Windows-based ODBC does not support certificate authentication.

NOTE

When connecting to the GaussDB server through ODBC, you can enable SSL to encrypt client-server communication. To do this, you must have the certificates and private key files required by the server and client. For instructions on obtaining these files, consult the relevant OpenSSL documentation and commands.

5.10.3 libpq

5.10.3.1 Connection Parameters

All connection parameters of libpq are constructed and passed as strings, regardless of whether the parameter type is host name, port number, password, or Boolean.

The following table describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 5-144 Connection parameters

Parameter	Description	Default Value
host	Name of the host to connect to. If the host name starts with a slash (/), UDS communications instead of TCP/IP communications are used. The value is the directory where the socket file is stored. If **host** is not specified, the default behavior is to connect to the UDS in the **/tmp** directory (or the socket directory specified during GaussDB installation). On a machine without a UDS, the default behavior is to connect to **localhost**. You can specify multiple host names by using a character string separated by commas (,). Multiple host names can be specified.	NULL
hostaddr	IP address of the host to connect to. The value is in standard IPv4 address format, for example, 172.28.40.9. If the machine supports IPv6, IPv6 address can also be used. If a non-null string is specified, TCP/IP communications are used. You can specify multiple IP addresses by using a character string separated by commas (,). Multiple IP addresses can be specified. Replacing **host** with **hostaddr** can prevent applications from querying host names, which may be important for applications with time limitations. However, a host name is required for GSSAPI or SSPI authentication methods. Therefore, the following rules are used: 1. If **host** is specified but **hostaddr** is not, a query for the host name will be executed. 2. If **hostaddr** is specified but **host** is not, the value of **hostaddr** is the server network address. If the host name is required for authentication, the connection attempt fails. 3. If both **host** and **hostaddr** are specified, the value of **hostaddr** is the server network address. The value of **host** is ignored unless it is required by authentication, in which case it is used as the host name. **NOTICE** • If **host** is not the server name in the network address specified by **hostaddr**, the authentication may fail. • If neither **host** nor **hostaddr** is specified, libpq will use a local UDS for connection. If the machine does not have a UDS, it will attempt to connect to **localhost**.	NULL

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
port	Port number of the host server, or the socket file name extension for UDS connections. You can specify multiple port numbers by using a character string separated by commas (,). Multiple port numbers can be specified.	NULL
user	Username of the connection. By default, the username is the same as the OS name of the user running the application.	Current OS username.
dbname	Database name. The default value is the same as the username.	Same as the **user** parameter.
password	Password to be used if the server requires password authentication.	NULL
connect_timeout	Maximum timeout period of the connection, in seconds. The value **0** or null indicates no timeout limit. You are advised not to set the connection timeout period to a value less than 2 seconds.	**0** (no timeout limit)
client_encoding	Client-side encoding format for the connection. In addition to the values accepted by the corresponding server options, you can use **auto** to determine the correct encoding from the current environment in the client (the *LC_CTYPE* environment variable in the Unix system).	Same as the server-side encoding format.
tty	Currently, this parameter does not take effect. (Previously, it was used to specify where the server's debugging output would be sent.)	NULL
options	Command-line options sent to the server when the connection starts. For example, **-c geqo=off** means setting the **geqo** parameter of the session to **off**.	NULL
application_name	Current user identity.	NULL
fallback_application_name	Fallback value for the **application_name** parameter. This value will be used when no value is given for **application_name** through a connection parameter or the environment variable *PGAPPNAME*. In a common tool program, if you set a default name but do not want the default name to be overwritten by the user, you can specify a backup value.	NULL

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
keepalives	Specifies whether to enable TCP keepalive for the client. By default, keepalive is enabled with a value of **1**. If keepalive is unnecessary, set this parameter to **0** to disable it. Ignore this parameter for UDS connections.	1
keepalives_idle	Duration of inactivity (in seconds) after which TCP will send a keepalive message to the server. When set to **0** or left unset, the system default will be applied. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored.	**0** (default value, indicating that the system TCP configuration is used)
keepalives_interval	Interval (in seconds) for retransmitting a TCP keepalive message that is not acknowledged by the server. When set to **0** or left unset, the system default will be applied. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored.	**0** (default value, indicating that the system TCP configuration is used)
keepalives_count	Number of times that a keepalive message can be retransmitted through TCP. When set to **0** or left unset, the system default will be applied. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored.	**0** (default value, indicating that the system TCP configuration is used)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
tcp_user_timeout	Maximum duration that transmitted data can remain unacknowledged before the TCP connection is forcibly closed. This parameter is intended for OSs that support the **TCP_USER_TIMEOUT** socket option. If this parameter is set to **0** or is left unset, the default value will be applied. Its unit is ms. Ignore this parameter for UDS connections.	**0** (default value, indicating that the system TCP configuration is used)
tcp_syn_retries	Number of times that the client can retransmit failed SYN packets during the three-way handshake phase. This parameter is available on OSs that support the socket option **TCP_SYNCNT**. When set to **0** or left unset, the system default will be applied. Ignore this parameter for UDS connections.	**0** (default value, indicating that the system TCP configuration is used)
rw_timeout	Read/Write timeout period of the client connection, in seconds. If set to the default value of **0**, no timeout period will be applied. If a timeout occurs on the libpq side and the connection is closed, the running services delivered by libpq to the database will be forcibly terminated. This capability is controlled by the GUC parameter **check_disconnect_query**. It is supported when the parameter is set to **on** and not supported when the parameter is set to **off**.	0

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
sslmode	Specifies whether to enable SSL encryption. • **disable**: SSL connections are not used. • **allow**: SSL connections are used when required by the database server. However, this mode does not verify the authenticity of the database server. • **prefer**: SSL connections are preferred if the database supports SSL. However, this mode does not verify the authenticity of the database server. • **require**: SSL connections are required. This mode only encrypts data but does not verify the authenticity of the database server. • **verify-ca**: SSL connections are required. Currently, Windows-based ODBC does not support certificate authentication. • **verify-full**: SSL connections are required. Currently, Windows-based ODBC does not support certificate authentication.	prefer
sslcompression	If this parameter is set to **1** (default value), the data transmitted over the SSL connection is compressed (this requires that the OpenSSL version be 0.9.8 or later). If this parameter is set to **0**, compression will be disabled (this requires OpenSSL 1.0.0 or later). If a connection without SSL is established, this parameter is ignored. If the OpenSSL version in use does not support this parameter, it will also be ignored. Compression takes up CPU time, but it increases throughput when the bottleneck is the network. If CPU performance is a limiting factor, disabling compression can improve response time and throughput.	**1** (SSL enabled)
sslcert	File name of the client SSL certificate. If no SSL connection is established, this parameter is ignored.	NULL
sslkey	Location of the key for the client certificate. It can specify a key obtained from an external "engine" (the engine is a loadable module of OpenSSL). The description of an external engine should consist of a colon-separated engine name and an engine-related key identifier. If no SSL connection is established, this parameter is ignored.	NULL
sslrootcert	File name of the SSL CA certificate. If the file exists, the system authenticates the server certificate issued by one of these authorities.	NULL
sslcrl	File name of the SSL certificate revocation list (CRL). If a certificate listed in this file exists, the server certificate authentication will be rejected.	NULL

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
requirepeer	OS user of the server, for example, **requirepeer=postgres**. When a UDS connection is established, if this parameter is set, the client checks whether the server process is running under the specified username at the beginning of the connection. If not, the connection will be interrupted by an error. This parameter can be used to provide server authentication similar to that of the SSL certificate on TCP/IP connections. (Note that if the UDS is in **/tmp** or another public writable location, any user can start a server for listening to the location. Use this parameter to ensure that you are connected to a server that is run by a trusted user.) This option is supported only on platforms that implement the peer authentication method.	NULL
krbsrvname	Kerberos service name used for GSSAPI authentication. For successful Kerberos authentication, this value must match the service name specified in the server configuration.	NULL
gsslib	GSS library used for GSSAPI authentication. It is used only in the Windows OS. If this parameter is set to **gssapi**, **libpq** is forced to use the GSSAPI library to replace the default SSPI for authentication.	NULL
service	Name of the service for which the additional parameter is used. It specifies a service name in **pg_service.conf** that holds the additional connection parameters. This allows the application to specify only one service name so that the connection parameters can be centrally maintained.	NULL
authtype	**authtype** is no longer used, so it is marked as a parameter not to be displayed. It is retained in an array so as not to reject the **conninfo** string from old applications that might still try to set it.	NULL
remote_nodename	Name of the remote node connected to the local node.	NULL
localhost	Localhost in a connection channel.	NULL
localport	Local port in a connection channel.	NULL
fencedUdfRPCMode	Specifies whether the fenced UDF RPC protocol uses UDSs or special socket file names. The default value is **0** (disabled), indicating that the UDS mode is used and the file type is .s.PGSQL.%d. To use the fenced UDF mode, set this parameter to **1** (enabled). In this case, the file type is .s.fencedMaster_unixdomain.	0

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
replication	Specifies whether the connection should use replication protocols instead of common protocols. This protocol is designed for PG replication connections and internal tools such as pg_basebackup, but it can also be used by third-party applications. The following values (case-insensitive) are supported: • **true**, **on**, **yes**, **1**, and **reply** Used for intra-cluster connections in physical replication mode. • **standby_cluster**, **hadr_main_standby**, and **hadr_standby_cn** Used for inter-cluster connections in physical replication mode. • **database** Used for connections to the database specified by **dbname** in logical replication mode. • **false**, **off**, **no**, **0**, and **NULL** (default value) Used for regular connections, which are the default behavior. In physical or logical replication mode, only simple query protocols can be used.	NULL
backend_version	Backend version passed to the remote end.	NULL
prototype	Current protocol level. The default value is **1**. Currently, this parameter is not in effect.	1
enable_ce	Specifies whether the client can connect to a fully-encrypted database. A default value of **0** indicates that the encrypted feature is disabled. If you need this query, change the value to **1** or **3**.	0
key_info	This parameter is used together with **enable_ce** to set parameters for accessing an external key manager in an encrypted database.	NULL
connection_info	**connection_info** is a JSON character string encompassing **driver_name**, **driver_version**, **driver_path**, and **os_user**. If it is not NULL, **connection_info** will be used, while **connectionExtraInfo** will be ignored. If it is NULL, a connection string specific to libpq will be generated. When **connectionExtraInfo** is set to **0**, **connection_info** will encompass only **driver_name** and **driver_version**.	0

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
connectionExtraInfo	Specifies whether to report the deployment path and process owner of the current driver to the database. By default, this feature is disabled with a value of **0**. To enable it, set this parameter to **1**. **NOTE** When this parameter is set to **1**, libpq will report the deployment path and process owner of the current driver to the database. This information will be recorded in the GUC parameter **connection_info** and can also be found in PG_STAT_ACTIVITY.	0
target_session_attrs	Type of host to connect to. The connection succeeds only when the actual host type matches the configured one. This parameter is verified only when multiple IP addresses are specified or the domain name corresponds to multiple IP addresses. The rules for setting **target_session_attrs** are as follows: • **any**: The connection is applicable to all types of hosts. • **read-write**: The connection is limited to hosts that are both readable and writable. • **read-only**: The connection is limited to read-only hosts. • **primary** (default value): The connection is limited to the primary node in the primary/standby system. • **standby**: The connection is limited to the standby node in the primary/standby system. • **prefer-standby**: The system first attempts to connect to the standby node in the primary/standby system. The **any** mode is used if the connection to all standby nodes on the **hosts** list fails.	primary
enable_host_cache	Specifies whether to enable the node cache functionality (**0**: disabled; **1**: enabled). It is disabled by default. **NOTE** The node cache does not take effect when a single IP address or **target_session_attrs=any** is configured in the connection string.	0
hostcache_expiration_time	After libpq attempts to connect to a host, the host status is recorded. This recorded host status remains trusted for the specified **hostcache_expiration_time**. Once this period expires, the recorded host status is no longer valid. The unit is seconds, with a default value of **10** and a valid range from 0 to **INT_MAX**. **NOTE** **hostcache_expiration_time** takes effect when **enable_host_cache** is set to **1**.	10

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Default Value
driver_type	Upper-layer driver type of libpq. The default value is **"GaussDB LIBPQ Driver"**. You are advised to retain the default value.	GaussDB LIBPQ Driver
force_no_truncation/ supportZeroCharacter	Specifies whether to force the use of the parameter length specified by the function caller (that is, the value of **paramLengths**) when certain database statement execution functions or asynchronous command processing functions pass parameters in text format (when **paramFormats** is **0**). The following rules apply for setting **force_no_truncation** or **supportZeroCharacter**: • **1**: enabled. The client's character set must be consistent with the database's character set. • **0** (default value): disabled. When **force_no_truncation** or **supportZeroCharacter** is enabled, the following database statement execution functions and asynchronous command processing functions are affected: • PQexecParams • PQexecParamsBatch • PQexecPrepared • PQexecPreparedBatch • PQsendQueryParams • PQsendQueryPrepared **NOTICE** This parameter must be enabled if you use the preceding database statement execution functions or asynchronous command processing functions to pass text parameter values containing the '\0' character and want to preserve the full parameter values (without truncation at '\0').	0

NOTICE

You are advised to set valid values for the preceding connection parameters. If invalid values are set, the connection may fail to be established, or the default values may be applied instead.

Initial Packet During Connection Establishment

After a socket is established during the connection process between the driver and the database, the first packet sent by the driver to the database, in key-value format, is called the initial packet. This packet includes the username (**user**), database name (**database**), certain GUC parameters (**replication**, **redistribution**, **backend_version**, **application_name**, **client_encoding**, **connect_timeout**,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

enable_full_encryption, **DateStyle**, **TimeZone**, **geqo**, **connection_info**, and **gaussdb.capacity_version**), and **options**. For detailed descriptions of these GUC parameters, refer to "Database GUC Parameters > GUC Parameters" in *Reference*.

NOTICE

- The **gaussdb.capacity_version** parameter indicates the version number of the compatibility capability set supported by the driver. It is used by the driver and the database to negotiate the compatibility feature set that will be applied during this connection. This is an internal parameter.
- After the database executes the following command, the values of the preceding GUC parameters in the initial packet will be updated to the values specified in the initial packet:
  ```
  reset all;
  ```

5.10.4 ECPG

5.10.4.1 Prepared Statements

The prepared statement mechanism is recommended when the condition parameters of an SQL statement are unknown at compile time or when the statement needs to be frequently executed in service logic. This mechanism significantly improves execution efficiency and prevents SQL injection.

- You can run the **PREPARE** command to create a prepared statement. For dynamic values that are only determined at runtime, use the placeholder "?".
  ```
  EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg_database WHERE oid = ?";
  ```
- If the statement returns a single row, the application can execute the statement by calling EXECUTE after PREPARE. The USING clause will supply the actual values for these placeholders.
  ```
  EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1;
  ```
- If the statement returns multiple rows, the application can use a cursor declared based on the statement. To bind input parameters, the cursor must be opened with the USING clause.
  ```
  EXEC SQL PREPARE stmt1 FROM "SELECT oid,datname FROM pg_database WHERE oid > ?";
  EXEC SQL DECLARE foo_bar CURSOR FOR stmt1;

  /* Break out of the while loop when the end of the result set is reached. */
  EXEC SQL WHENEVER NOT FOUND DO BREAK;
  EXEC SQL OPEN foo_bar USING 100;
  ...
  while (1)
  {
     EXEC SQL FETCH NEXT FROM foo_bar INTO :dboid, :dbname;
     ...
  }
  EXEC SQL CLOSE foo_bar;
  ```
- When a prepared statement is no longer needed, it should be deallocated.
  ```
  EXEC SQL DEALLOCATE PREPARE name;
  ```

5.10.4.2 Data Types and Host Variables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.10.4.2.1 Overview

This section details how to use host variables to efficiently and securely transfer data between C programs and embedded SQL statements.

In embedded SQL-C programs, the C language serves as the host language. Variables defined in C code and referenced in EXEC SQL statements are collectively referred to as host variables. They act as a primary interface between the C memory space and data stored in the database kernel.

The core of data interaction lies in matching C's native data types with GaussDB data types. To prevent precision loss, type conflicts, and memory overflow during data transfer, it is crucial to understand the type mapping mechanism. When declaring host variables, you must select appropriate C data types (including primitive types, structures, unions, and arrays) based on the attributes of underlying GaussDB fields. This ensures accurate data alignment during precompilation.

When referencing host variables in SQL statements, there is no need to manually concatenate data into strings. Instead, simply prefix the variable names with a colon (:). ECPG will automatically recognize and process these references. This parameterized reference approach not only simplifies development but also fundamentally enhances programs' ability to defend against SQL injection attacks.

Example:

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

This statement references two C variables, v1 and v2, demonstrating that host variables can be flexibly combined with SQL literals (such as 'foo') within the same SQL statement. This offers significant flexibility in statement construction.

5.10.4.2.2 DECLARE Section

To enable bidirectional data transfer between embedded SQL-C programs and databases—such as passing parameters from C programs to SQL statements or returning query results to C programs—you must define relevant C variables in a dedicated DECLARE section. This section informs ECPG which variables will be used for SQL interaction, allowing it to accurately identify the memory addresses and data types of these variables.

The DECLARE section must start with the **EXEC SQL BEGIN DECLARE SECTION;** directive and end with **EXEC SQL END DECLARE SECTION;**. Between those lines, you can declare variables using standard C syntax:

```
EXEC SQL BEGIN DECLARE SECTION;
int x = 4;
char foo[16], bar[16];
EXEC SQL END DECLARE SECTION;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- The type of the host variables declared between the start and end of the marked section must be one of the supported data types. For details, see **Table 5-145**.
- For simple single variables, you can omit the full DECLARE section by using a directive prefix for implicit declaration, for example, **EXEC SQL int i = 4**.
- Variables not used within SQL commands may be declared outside the DECLARE section. Only variables declared inside the DECLARE section are usable in SQL statements.
- If a host variable involves a user-defined structure or union, its complete type definition must also be listed inside the DECLARE section. Otherwise, ECPG cannot resolve its underlying memory structure.

5.10.4.2.3 Retrieving Query Results

To retrieve the results of a query, embedded SQL provides special variants of the usual commands **SELECT** and **FETCH**. These commands have a special INTO clause that specifies which host variables the retrieved values are to be stored in. **SELECT** is used for a query that returns only a single row, and **FETCH** is used for a query that returns multiple rows, using a cursor.

- Here is an example for using the **SELECT** command:

```
/*
* Assume that the test table already exists.
 * CREATE TABLE test (a int, b varchar(50));
 */
    EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    VARCHAR v2;
    EXEC SQL END DECLARE SECTION;

    ...

    EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;
```

  The INTO clause appears between the SELECT list and the FROM clause. The number of elements in the SELECT list and the list after INTO (also called the target list) must be equal.

- Here is an example for using the **FETCH** command:

```
EXEC SQL BEGIN DECLARE SECTION;
int v1;
VARCHAR v2;
EXEC SQL END DECLARE SECTION;
...

EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;
...
do
{
   ...
   EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
   ...
} while (...);
```

  The INTO clause appears after all SQL clauses.

5.10.4.2.4 Data Type Mapping

When ecpg applications exchange values between the GaussDB server and the C program, such as when retrieving query results from the server or executing SQL statements with input parameters, the values need to be converted between GaussDB data types and host language variable types (C language data types,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

concretely). In this respect, there are two data types available: Simple GaussDB data types, such as integer and text, can be directly read and written by the application. Other GaussDB data types, such as timestamp and numeric, can only be accessed through special library functions. For details, see **ECPG API Reference**.

NOTE

- Currently, only basic data types of the C language can be used or combined. The string data type in the C++ language cannot be used as the host variable type.
- Currently, ECPG maps only common data types of SQL. For details about the supported data types, see **Table 5-145**.

Table 5-145 Mapping between GaussDB data types and C variable types

GaussDB Data Type	Host Variable Type
smallint	short
integer	int
bigint	long long int
boolean	boolean
character(n), varchar(n), text	char[n+1], VARCHAR[n+1]
double precision	double
real	float
smallserial	short
serial	int
bigserial	long long int
oid	unsigned int
name	char[NAMEDATALEN]
date	date [a] **NOTE** [a] indicates that this is special data type, and its access method is detailed in **Accessing Special Data Types**.
timestamp	timestamp [a]
interval	interval [a]
decimal	decimal [a]
numeric	numeric [a]

5.10.4.2.5 Handling Character Strings

To handle SQL character string data types, such as varchar and text, there are two possible methods to declare the host variables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Use char[] (a char string), which is the most common method to handle character data in C programs.

```
EXEC SQL BEGIN DECLARE SECTION;
   char str[50];
EXEC SQL END DECLARE SECTION;
```

  Note that you have to take care of the length yourself. If you use this host variable as the target variable of a query which returns a string with more than 49 bytes, a buffer overflow occurs.

- Use the VARCHAR type, which is a special type provided by ECPG. The definition on an array of the VARCHAR type is converted into the struct type. A declaration like:

```
VARCHAR var[180];
```

  is converted into:

```
struct varchar_var
{
   int len;
   char arr[180];
} var;
```

  To store a string in a VARCHAR host variable, the host variable has to be declared with the length including the zero byte ('\0') terminator. The **arr** field holds the string including the terminating zero byte. The **len** field holds the length of the string stored in **arr**. The terminator is excluded during length calculation. When a host variable is used as input for a query, if **strlen(arr)** and **len** are different, the shorter one is used.

  NOTE

  - VARCHAR can be written in upper or lower case, but not in mixed case.
  - char and VARCHAR host variables can also hold values of other SQL types, which will be stored in their string forms.

5.10.4.2.6 Host Variables with Non-primitive Types

Non-primitive host variables can be arrays, typedefs, structures, and pointers.

Arrays

There are two use cases for arrays as host variables.

1. Store text strings in char[] or VARCHAR[].
2. Retrieve multiple rows from a query result without using a cursor.

Without an array, to process a query result consisting of multiple rows, **it is required to use a cursor and the FETCH command**. But with array host variables, multiple rows can be retrieved at once. The length of the array has to be defined to be able to accommodate all rows; otherwise, a buffer overflow will likely occur.

The following is an example of scanning the pg_database system catalog and displays the OIDs and names of all available databases.

Prerequisites: The **testdb** database has been created.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
   EXEC SQL BEGIN DECLARE SECTION;
```

```
    int dbid[8];
    char dbname[8][16];
    int i;
    EXEC SQL END DECLARE SECTION;

    memset(dbname, 0, sizeof(char)* 16 * 8);
    memset(dbid, 0, sizeof(int) * 8);

   /* Connect to testdb. */
    EXEC SQL CONNECT TO testdb;

    /* Retrieve multiple rows into arrays at once. */
    EXEC SQL SELECT oid,datname INTO :dbid, :dbname FROM pg_database;
    for (i = 0; i < 8; i++)
        printf("oid=%d, dbname=%s\n", dbid[i], dbname[i]);
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

This example shows the following result (the exact values depend on local circumstances):

```
oid=1, dbname=template1
oid=11510, dbname=template0
oid=11511, dbname=postgres
oid=313780, dbname=testdb
oid=0, dbname=
oid=0, dbname=
oid=0, dbname=
```

Structures

A structure whose member names match the column names of a query result can be used to retrieve multiple columns at once. The structure enables handling multiple column values in a single host variable.

The following example retrieves OIDs, names, and sizes of the available databases from the pg_database system catalog by using the pg_database_size() function. In this example, a structure variable with members whose names match each column in the SELECT result is used to retrieve one result row without putting multiple host variables in the FETCH statement.

```
    EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
        int oid;
        char datname[65];
        long long int size;
    } dbinfo_t;

    dbinfo_t dbval;
    EXEC SQL END DECLARE SECTION;
    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
    EXEC SQL OPEN cur1;

    /* Break out of the while loop when the end of the result set is reached. */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* Fetch multiple columns into one structure. */
        EXEC SQL FETCH FROM cur1 INTO :dbval;
        /* Print members of the structure. */
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, dbval.size);
}
EXEC SQL CLOSE cur1;
```

This example shows the following result (the exact values depend on local circumstances):

```
oid=1, datname=template1, size=4324580
oid=11510, datname=template0, size=4243460
oid=11511, datname=postgres, size=4324580
oid=313780, datname=testdb, size=8183012
```

Structure host variables "absorb" as many columns in the query result as the structure columns. Additional columns can be allocated to other host variables. The above program could also be restructured like this, with the *size* variable outside the structure:

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct
{
    int oid;
    char datname[65];
} dbinfo_t;

dbinfo_t dbval;
long long int size;
EXEC SQL END DECLARE SECTION;

memset(&dbval, 0, sizeof(dbinfo_t));

EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
EXEC SQL OPEN cur1;

/* Break out of the while loop when the end of the result set is reached. */
EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
    /* Fetch multiple columns into one structure. */
    EXEC SQL FETCH FROM cur1 INTO :dbval, :size;
    /* Print members of the structure. */
    printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, size);
}

EXEC SQL CLOSE cur1;
```

Typedefs

Use the typedef keyword to map new types to existing types.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef char mychartype[40];
typedef long serial_t;
EXEC SQL END DECLARE SECTION;
```

Note that you could also use:

```
EXEC SQL TYPE serial_t IS long;
```

This declaration does not need to be part of a DECLARE section.

Pointers

You can declare pointers to the most common types.

```
EXEC SQL BEGIN DECLARE SECTION;
int   *intp;
char **charp;
EXEC SQL END DECLARE SECTION;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5.10.4.2.7 Accessing Special Data Types

ECPG supports the numeric, decimal, date, timestamp, and interval types. These data types cannot be mapped to primitive host variable types, because they have a complex internal structure. Applications deal with these types by declaring host variables in special types and accessing them using functions in the pgtypes library. For details about the functions in the pgtypes library, see **ECPG API Reference**.

- Timestamp and date

  First, the program has to include the header file for the timestamp type.

  ```
  #include <pgtypes_timestamp.h>
  ```

  Next, declare a host variable of the timestamp type in the DECLARE section.

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    timestamp ts;
  EXEC SQL END DECLARE SECTION;
  ```

  After a value is read into the host variable, process it using pgtypes library functions. In the following example, the **timestamp** value is converted into a text form with the PGTYPEStimestamp_to_asc() function:

  ```
  EXEC SQL SELECT now()::timestamp INTO :ts;
  printf("ts = %s\n", PGTYPEStimestamp_to_asc(ts));
  ```

  This example shows the following result:

  ```
  ts = 2022-06-27 18:03:56.949343
  ```

  In addition, the date type can be handled in the same way. The program has to include the header file **pgtypes_date.h**, declare a host variable as the date type, and convert a date value into a text form using the PGTYPEdata_to_asc() function.

- Interval

  The handling of the interval type is also similar to the timestamp and date types. It is required, however, to allocate some memory space on the heap for an interval type variable.

  Example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <pgtypes_interval.h>
  int main(void)
  {
    EXEC SQL BEGIN DECLARE SECTION;
    interval *in;
    EXEC SQL END DECLARE SECTION;

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb;
    in = PGTYPESinterval_new();
    EXEC SQL SELECT '1 min'::interval INTO :in;
    printf("interval = %s\n", PGTYPESinterval_to_asc(in));
    PGTYPESinterval_free(in);
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
  }
  ```

  The result is as follows:

  ```
  interval = @ 1 min
  ```

- Numeric and decimal

  The handling of the numeric and decimal types is similar to the interval type: It requires defining a pointer, allocating some memory space on the heap, and accessing the variable using the pgtypes library functions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
EXEC SQL WHENEVER SQLERROR STOP;
int main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    numeric *num;
    numeric *num2;
    decimal *dec;
    EXEC SQL END DECLARE SECTION;

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb;

    num = PGTYPESnumeric_new();
    dec = PGTYPESdecimal_new();

    EXEC SQL SELECT 12.345::numeric(4,2), 23.456::decimal(4,2) INTO :num, :dec;
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 0));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 1));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 2));

    /* Convert decimal to numeric to show a decimal value. */
    num2 = PGTYPESnumeric_new();
    PGTYPESnumeric_from_decimal(dec, num2);
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 0));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 1));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 2));
    PGTYPESnumeric_free(num2);
    PGTYPESdecimal_free(dec);
    PGTYPESnumeric_free(num);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

The result is as follows:

```
numeric = 12
numeric = 12.4
numeric = 12.35
decimal = 23
decimal = 23.5
decimal = 23.46
```

5.10.4.2.8 Handling Non-primitive SQL Data Types

This section contains information on how to handle nonscalar and user-defined SQL-level data types in ECPG applications. Note that this is distinct from the handling of host variables of non-primitive types described in **Host Variables with Non-primitive Types**.

- Arrays

  Multi-dimensional SQL-level arrays are not directly supported in ECPG. One-dimensional SQL-level arrays can be mapped to C array host variables. However, when creating a statement, ECPG does not know the types of the columns, so that it cannot check if a C array is input into a corresponding SQL-level array. When processing the output of an SQL statement, ECPG has to check if both are arrays.

  If a query accesses elements of an array separately, a host variable with a type that can be mapped to the element type should be used. For example, if a column type is integer array, a host variable of the int type can be used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Likewise, if the element type is varchar or text, a host variable of the char[] or VARCHAR[] type can be used.

Example:

```
CREATE TABLE t3 (
    ii integer[]
);
testdb=> SELECT * FROM t3;
     ii
-------------
 {1,2,3,4,5}
(1 row)
```

The following example program retrieves the 4th element of the array and stores it into a host variable of the int type:

```
    EXEC SQL BEGIN DECLARE SECTION;
    int ii;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH FROM cur1 INTO :ii ;
        printf("ii=%d\n", ii);
    }
    EXEC SQL CLOSE cur1;
```

This example shows the following result:

```
ii=4
```

To map multiple array elements to an array-type host variable, each element of the array column and host variable array have to be managed separately, for example:

```
    EXEC SQL BEGIN DECLARE SECTION;
    int ii_a[8];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4] FROM t3;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while (1)
    {
        EXEC SQL FETCH FROM cur1 INTO :ii_a[0], :ii_a[1], :ii_a[2], :ii_a[3];
        ...
    }
```

Note that it would not work correctly in this case, because you cannot map an array type column to an array host variable directly.

```
    EXEC SQL BEGIN DECLARE SECTION;
    int ii_a[8];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* Error */
        EXEC SQL FETCH FROM cur1 INTO :ii_a;
        ...
    }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Composite types

  Composite types are not directly supported in ECPG. For example, in struct, you cannot declare member variables as the date type. However, you can access each attribute in the composite types separately or use the external string representation.

  In the following example, each attribute can be accessed separately:

```
CREATE TYPE comp_t AS (intval integer, textval varchar(32));
CREATE TABLE t4 (compval comp_t);
INSERT INTO t4 VALUES ( (256, 'GaussDB') );
```

  The following program retrieves each attribute of the comp_t type separately:

```
EXEC SQL BEGIN DECLARE SECTION;
int intval;
varchar textval[33];
EXEC SQL END DECLARE SECTION;

/* Put each element of the composite type column in the SELECT list. */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
    /* Fetch each element of the composite type column into host variables. */
    EXEC SQL FETCH FROM cur1 INTO :intval, :textval;
    printf("intval=%d, textval=%s\n", intval, textval.arr);
}
EXEC SQL CLOSE cur1;
```

  The host variables storing values in the **FETCH** command can be gathered into one structure. For more details about the host variables in the structure form, see **Handling Character Strings**. In the following example, the two host variables, *intval* and *textval*, become members of the comp_t structure, and the structure is specified in the **FETCH** command:

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct
{
    int intval;
    varchar textval[33];
} comp_t;
comp_t compval;
EXEC SQL END DECLARE SECTION;

/* Put each element of the composite type column in the SELECT list. */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
while (1)
{
    /* Put all values in the SELECT list into one structure. */
    EXEC SQL FETCH FROM cur1 INTO :compval;
    printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
}
EXEC SQL CLOSE cur1;
```

  Although a structure is used in the **FETCH** command, the attribute names in the SELECT clause are specified one by one. This can be enhanced by using an asterisk (*) to ask for all attributes of the composite type value, for example:

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    {
        /* Put all values in the SELECT list into one structure. */
        EXEC SQL FETCH FROM cur1 INTO :compval;
        printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
    }
...
```

In this way, composite types can be mapped into structures, even though ECPG does not understand the composite type itself.

- User-defined base types

  When ECPG uses host variables to store query results, only the data types provided by ECPG are supported. User-defined data types are not supported. Data types created using CREATE TYPE cannot be mapped using host variables.

  However, you can use external string representations and host variables of the char[] or VARCHAR[] type to handle user-defined types.

  The external string representation of the data type complex is (%lf,%lf), which is defined in the complex_in() function. The following example inserts complex type values **(1,1)** and **(3,3)** into columns **a** and **b**, and selects them from the table after that:

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  varchar a[64];
  varchar b[64];
  EXEC SQL END DECLARE SECTION;
  EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)');
  EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex;
  EXEC SQL OPEN cur1;

  EXEC SQL WHENEVER NOT FOUND DO BREAK;

  while (1)
  {
     EXEC SQL FETCH FROM cur1 INTO :a, :b;
     printf("a=%s, b=%s\n", a.arr, b.arr);
  }
  EXEC SQL CLOSE cur1;
  ```

  This example shows the following result:

  ```
  a=(1,1), b=(3,3)
  ```

5.10.4.3 Executing Dynamic SQL Statements

5.10.4.3.1 Overview

In many cases, the particular SQL statements that an application has to execute are known at the time the application is written.

In some cases, however, the SQL statements are composed at run time or provided by an external source. In these cases, you cannot embed the SQL statements directly into the C source code, but there is a facility that allows you to call the provided dynamic SQL statements through a string variable.

5.10.4.3.2 Executing a Statement Without a Result Set

Here is an example for the **EXECUTE IMMEDIATE** command:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;
EXEC SQL EXECUTE IMMEDIATE :stmt;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

EXECUTE IMMEDIATE can be used for SQL statements that do not return a result set (for example, DDL, INSERT, UPDATE, and DELETE). You cannot execute statements that retrieve data (for example, SELECT) this way.

5.10.4.3.3 Executing a Statement with Input Parameters

Prepare an SQL statement that includes a question mark (?) as a parameter placeholder.

Use the EXECUTE statement along with the USING clause to execute that statement and pass the parameter, like the following:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;
/* Prepare a statement for execution. */
EXEC SQL PREPARE mystmt FROM :stmt;
...
/* Single quotation marks are valid characters. When dealing with a character string, use double quotation marks. */
EXEC SQL EXECUTE mystmt USING 42, 'foobar';

/* When you no longer need a prepared statement, deallocate it. */
EXEC SQL DEALLOCATE PREPARE name;
```

5.10.4.3.4 Executing a Statement with a Result Set

EXECUTE can be used to execute an SQL statement that generates a separate result set.

To save the result, you need to include the INTO clause, as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
int v1, v2;
VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
...
EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

NOTE

The EXECUTE statement supports the INTO and USING clauses.

If a query may return multiple result rows, use a cursor (see **Using Cursors** for more details), as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char dbaname[128];
char datname[128];
char *stmt = "SELECT u.usename as dbaname, d.datname "
             "  FROM pg_database d, gs_user u "
             "  WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO testdb AS con1 USER testuser;

EXEC SQL PREPARE stmt1 FROM :stmt;

EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
EXEC SQL OPEN cursor1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    EXEC SQL FETCH cursor1 INTO :dbaname,:datname;
    printf("dbaname=%s, datname=%s\n", dbaname, datname);
}
EXEC SQL CLOSE cursor1;

EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
```

5.10.4.4 Error Handling

5.10.4.4.1 Overview

There are two non-exclusive methods to handle errors and warnings in an embedded SQL program:

- Use the **WHENEVER** command to set callbacks to handle warnings and errors. For details, see **Setting Callbacks**.
- Obtain the detailed errors or warnings through *sqlca* to handle the errors or warnings. For details, see **sqlca**.

5.10.4.4.2 Setting Callbacks

One simple method to catch errors and warnings is to set a specific action to be executed whenever a particular condition occurs. In general:

```
EXEC SQL WHENEVER condition action;
```

condition can be one of the following:

- **SQLERROR**: The specified action is called whenever an error occurs during the execution of an SQL statement.
- **SQLWARNING**: The specified action is called whenever a warning occurs during the execution of an SQL statement.
- **NOT FOUND**: The specified action is called whenever an SQL statement retrieves or affects zero rows.

action can be one of the following:

- **CONTINUE**: ignores the callback error condition and continues with the execution. This is the default value and is usually used to stop a break condition.
- **GOTO** *label*/**GO TO** *label*: jumps to the specified label (using a C goto statement).
- **SQLPRINT**: prints a message to the standard error.
- **STOP**: calls exit(1) to terminate the program.
- **DO BREAK**: executes the C statement BREAK. This should only be used in loops or switch statements.

Example:

```
/* Print a message when a warning occurs. */
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
/* Terminate the program when an error occurs. */
EXEC SQL WHENEVER SQLERROR STOP;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- EXEC SQL WHENEVER is a directive of the SQL preprocessor, not a C statement. The error or warning actions that it sets apply to all embedded SQL statements that appear below the point where the handler is set, unless a different action was set for the same condition between the first EXEC SQL WHENEVER and the SQL statement causing the condition. Therefore, neither of the following C programs will have the expected effect:

```
/* Error */
void func()
{
    ...
    if (verbose) {
        EXEC SQL WHENEVER SQLWARNING SQLPRINT;
    }
    ...
    EXEC SQL SELECT ...;
    ...
}
/* Error */
void func()
{
    ...
    set_error_handler();
    ...
    EXEC SQL SELECT ...;
    ...
}
static void set_error_handler(void)
{
    EXEC SQL WHENEVER SQLERROR STOP;
}
```

- DO BREAK can only be used in the while, for, and switch scenarios. Upon the completion of its execution, you need to run the CONTINUE statement to ignore the callback error condition.

5.10.4.4.3 sqlca

The embedded SQL API provides a global variable *sqlca* (short for SQL communication area). *sqlca* covers both warnings and errors. If multiple warnings or errors occur during the execution of a statement, then *sqlca* will only contain information about the last one. In a multithreaded program, every thread automatically gets its own copy of *sqlca*.

The data structure is as follows:

```
struct
{
    char sqlcaid[8];
    long sqlabc;
    long sqlcode;
    struct
    {
        int sqlerrml;
        char sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlstate[5];
} sqlca;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If no error occurred in the last SQL statement, **sqlca.sqlcode** will be **0** and **sqlca.sqlstate** will be **"00000"**. If a warning or error occurred, then **sqlca.sqlcode** will be negative and **sqlca.sqlstate** will be different from **"00000"**. For details about the values of **SQLSTATE** and **SQLCODE**, see **SQLSTATE and SQLCODE**.

If the last SQL statement was successful, then **sqlca.sqlerrd[1]** contains the OID of the processed row, and **sqlca.sqlerrd[2]** contains the number of processed or returned rows. In case of an error or warning, **sqlca.sqlerrm.sqlerrmc** will contain a string that describes the error. The **sqlca.sqlerrm.sqlerrml** field contains the length of the error message, that is, the result of **strlen()**. Note that some messages are too long to fit in the fixed-size sqlerrmc array; they will be truncated.

In case of a warning, **sqlca.sqlwarn[2]** is set to **W**.

The **sqlcaid**, **sqlabc**, **sqlerrp** fields, and the remaining elements of **sqlerrd** and **sqlwarn** currently contain no useful information.

Example:

```
/* Combine WHENEVER and sqlca for error handling. */
EXEC SQL WHENEVER SQLERROR SQLCALL print_sqlca();

void print_sqlca()
{
    fprintf(stderr, "==== sqlca ====\n");
    fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);
    fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.sqlerrml);
    fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.sqlerrmc);
    fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.sqlerrd[0],sqlca.sqlerrd[1],sqlca.sqlerrd[2],
                                                          sqlca.sqlerrd[3],sqlca.sqlerrd[4],sqlca.sqlerrd[5]);
    fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.sqlwarn[0], sqlca.sqlwarn[1],
sqlca.sqlwarn[2],
                                                          sqlca.sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],
                                                          sqlca.sqlwarn[6], sqlca.sqlwarn[7]);
    fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);
    fprintf(stderr, "===============\n");
}
```

The result could look as follows (here an error due to a misspelled table name):

```
==== sqlca ====
sqlcode: -400
sqlerrm.sqlerrml: 49
sqlerrm.sqlerrmc: relation "pg_databasep" does not exist on line 38
sqlerrd: 0 0 0 0 0 0
sqlwarn: 0 0 0 0 0 0 0 0
sqlstate: 42P01
===============
```

5.10.4.4.4 SQLSTATE and SQLCODE

SQLSTATE is a five-character array. The five characters contain digits or upper-case letters that represent codes of various error and warning conditions. SQLSTATE is organized hierarchically: the first two characters denote a class of errors, while the last three characters indicate a specific condition within that class. For example, a successful state is indicated by the code 00000.

SQLCODE is a simple integer that reflects the execution status of SQL statements. Specifically:

- A value of 0 indicates success.
- A positive value indicates success with additional information. The SQL standard only defines the positive value +100, which indicates that the last command returned or affected zero rows.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- A negative value indicates an error.

It is worth mentioning that the SQL standard does not provide definitions for negative values. Instead, negative values are typically defined by the database system to indicate different error conditions.

NOTE

- ECPG includes these new SQLSTATE codes for embedded SQL statements: 22002, 07001, 07002, 07006, 07009, 33000, 42601, 42804, 42P03, YE000, and YE001. Other SQLSTATE codes are inherited from the kernel.
- An SQLSCODE of **-400** indicates that ECPG has detected an error from the kernel server, and the corresponding SQLSTATE is the same as that defined by the kernel.

Table 5-146 Mapping between SQLSTATE and SQLCODE

SQLCODE Value	SQLSTATE Value	Description
0 (ECPG_NO_ERROR)	SQLSTATE 00000	Indicates no error.

100 (ECPG_NOT_FOUND)	SQLSTATE 02000	This is a harmless condition indicating that the last command retrieved or processed zero rows, or that you are at the end of the cursor.
		When processing a cursor in a loop, you can use this code as a way to detect when to abort the loop, like this:
		`while (1)`
		`{`
		`  EXEC SQL FETCH ... ;`
		`  if (sqlca.sqlcode == ECPG_NOT_FOUND)`
		`  break;`
		`}`
		But WHENEVER NOT FOUND DO BREAK effectively does this internally, so there is usually no advantage in writing this out explicitly.

-12 (ECPG_OUT_OF_MEMORY)	SQLSTATE YE001	Indicates that your virtual memory is exhausted. The numeric value is defined as **-ENOMEM**.
-200 (ECPG_UNSUPPORTED)	SQLSTATE YE000	Indicates that the preprocessor has generated something that the library does not know about.
-201 (ECPG_TOO_MANY_ARGUMENTS)	SQLSTATE 07001 or 07002	Indicates that the command specified more host variables than the command expected.
-202 (ECPG_TOO_FEW_ARGUMENTS)	SQLSTATE 07001 or 07002	Indicates that the command specified fewer host variables than the command expected.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQLCODE Value	SQLSTATE Value	Description
-203 (ECPG_TOO_MANY_MATCHES)	SQLSTATE 21000	Indicates that a query has returned multiple rows, but the statement was only prepared to store one result row.
-204 (ECPG_INT_FORMAT)	SQLSTATE 42804	The host variable is of the int type, but the data in the database is of a different type and contains a value that cannot be interpreted as an int value. The library uses strtol() for this conversion.
-205 (ECPG_UINT_FORMAT)	SQLSTATE 42804	The host variable is of the unsigned int type, but the data in the database is of a different type and contains a value that cannot be interpreted as an unsigned int value. The library uses strtoul() for this conversion.
-206 (ECPG_FLOAT_FORMAT)	SQLSTATE 42804	The host variable is of the float type, but the data in the database is of a different type and contains a value that cannot be interpreted as a float value. The library uses strtod() for this conversion.
-207 (ECPG_NUMERIC_FORMAT)	SQLSTATE 42804	The host variable is of the numeric type, but the data in the database is of a different type and contains a value that cannot be interpreted as a numeric value.
-208 (ECPG_INTERVAL_FORMAT)	SQLSTATE 42804	The host variable is of the interval type, but the data in the database is of a different type and contains a value that cannot be interpreted as an interval value.
-209 (ECPG_DATE_FORMAT)	SQLSTATE 42804	The host variable is of the date type, but the data in the database is of a different type and contains a value that cannot be interpreted as a date value.
-210 (ECPG_TIMESTAMP_FORMAT)	SQLSTATE 42804	The host variable is of the timestamp type, but the data in the database is of a different type and contains a value that cannot be interpreted as a timestamp value.
-211 (ECPG_CONVERT_BOOL)	SQLSTATE 42804	The host variable is of the Boolean type, but the data in the database is neither 't' nor 'f'.
-212 (ECPG_EMPTY)	SQLSTATE YE000	The statement sent to the SQL server was empty. (This cannot normally happen in an embedded SQL program, so it might point to an internal error.)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQLCODE Value	SQLSTATE Value	Description
-213 (ECPG_MISSING_INDICATOR)	SQLSTATE 22002	A null value was returned, and no null indicator variable was supplied.
-214 (ECPG_NO_ARRAY)	SQLSTATE 42804	An ordinary variable was used in a place that requires an array.
-215 (ECPG_DATA_NOT_ARRAY)	SQLSTATE 42804	The database returned an ordinary variable in a place that requires array value.
-216 (ECPG_ARRAY_INSERT)	SQLSTATE 42804	The value could not be inserted into the array.
-220 (ECPG_NO_CONN)	SQLSTATE 08003	The program tried to access a connection that does not exist.
-221 (ECPG_NOT_CONN)	SQLSTATE YE000	The program tried to access a connection that does exist but is not open. (This is an internal error.)
-230 (ECPG_INVALID_STMT)	SQLSTATE 26000	The statement you are trying to use has not been prepared.
-239 (ECPG_INFORMIX_DUPLICATE_KEY)	SQLSTATE 23505	Duplicate key error, violation of the UNIQUE constraint.
-240 (ECPG_UNKNOWN_DESCRIPTOR)	SQLSTATE 33000	The descriptor specified was not found. The statement you are trying to use has not been prepared.
-241 (ECPG_INVALID_DESCRIPTOR_INDEX)	SQLSTATE 07009	The descriptor index specified was out of range.
-242 (ECPG_UNKNOWN_DESCRIPTOR_ITEM)	SQLSTATE YE000	An invalid descriptor item was requested. (This is an internal error.)
-243 (ECPG_VAR_NOT_NUMERIC)	SQLSTATE 07006	During the execution of a dynamic statement, the database returned a numeric value and the host variable was not numeric.
-244 (ECPG_VAR_NOT_CHAR)	SQLSTATE 07006	During the execution of a dynamic statement, the database returned a non-numeric value and the host variable was numeric.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQLCODE Value	SQLSTATE Value	Description
-284 (ECPG_INFORMIX_SUBSELECT_NOT_ONE)	SQLSTATE 21000	A result of the subquery is not single row.
-400 (ECPG_PGSQL)	-	Some error caused by the SQL server. This message contains the error message from the SQL server.
-401 (ECPG_TRANS)	SQLSTATE 08007	The SQL server signaled that the transaction cannot be started, committed, or rolled back.
-402 (ECPG_CONNECT)	SQLSTATE 08001	The connection attempt to the database did not succeed.
-403 (ECPG_DUPLICATE_KEY)	SQLSTATE 23505	Duplicate key error, violation of the UNIQUE constraint.
-404 (ECPG_SUBSELECT_NOT_ONE)	SQLSTATE 21000	A result of the subquery is not single row.
-602 (ECPG_WARNING_UNKNOWN_PORTAL)	SQLSTATE 34000	An invalid cursor name was specified.
-603 (ECPG_WARNING_IN_TRANSACTION)	SQLSTATE 25001	A transaction is in progress.
-604 (ECPG_WARNING_NO_TRANSACTION)	SQLSTATE 25P01	There is no active (in-progress) transaction.
-605 (ECPG_WARNING_PORTAL_EXISTS)	SQLSTATE 42P03	An existing cursor name was specified.

5.10.4.5 Using Library Functions

ECPGdebug(int on, FILE *stream)

If the first parameter in this function is not **0**, the debug log feature will be enabled. The second parameter determines the standard output stream for the logs to be printed. Debug logs are executed on the standard output stream. The logs record all input SQL statements and results from the GaussDB server.

Example:

```
#include <stdio.h>
#include <stdlib.h>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
#include <string.h>
#include "sqlca.h"

int main()
{
    ECPGdebug(1, stderr);

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb;
    EXEC SQL SET AUTOCOMMIT TO ON;
    EXEC SQL CREATE TABLE T1(a int);
    return (0);
}
```

Result (**[23893]** is a randomly generated thread ID):

```
[23893]: ECPGdebug: set to 1
[23893]: ECPGconnect: opening database testdb on <DEFAULT> port <DEFAULT>
[23893]: ECPGsetcommit on line 12: action "on"; connection "testdb"
[23893]: ecpg_execute on line 13: query: create table T1 ( a int ); with 0 parameter(s) on connection testdb
[23893]: ecpg_execute on line 13: using PQexec
[23893]: ecpg_execute on line 13: OK: CREATE TABLE
```

ECPGget_PGconn(const char *connection_name)

Returns the database connection handle identified by the given name. When *connection_name* is set to **NULL**, the function will return the current connection handle. If no connection handle can be identified, the function will return **NULL**.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
    ECPGdebug(1, stderr);

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb as con1;
    EXEC SQL SET AUTOCOMMIT TO ON;
    EXEC SQL DROP TABLE IF EXISTS T1;
    PGconn     *conn;
    conn = ECPGget_PGconn("con1");
    printf("conn = %p\n", conn);
    conn = ECPGget_PGconn(NULL);
    printf("conn = %p\n", conn);
    EXEC SQL CREATE TABLE T1(a int);
    return (0);
}
```

Result (**[29709]** is a randomly generated thread ID, and the connected address is also randomly generated, resulting in non-fixed results):

```
[29709]: ECPGdebug: set to 1
[29709]: ECPGconnect: opening database testdb on <DEFAULT> port <DEFAULT>
[29709]: ECPGsetcommit on line 12: action "on"; connection "con1"
[29709]: ecpg_execute on line 13: query: drop table if exists T1; with 0 parameter(s) on connection con1
[29709]: ecpg_execute on line 13: using PQexec
[29709]: ecpg_execute on line 13: OK: DROP TABLE
conn = 0x14daf70
conn = 0x14daf70
[29709]: ecpg_execute on line 19: query: create table T1 ( a int ); with 0 parameter(s) on connection con1
[29709]: ecpg_execute on line 19: using PQexec
[29709]: ecpg_execute on line 19: OK: CREATE TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

ECPGtransactionStatus(const char *connection_name)

Returns the current transaction status of the connection specified in **connection_name**. The possible return values include:

```
PQTRANS_IDLE,    /* Connection idle */
PQTRANS_ACTIVE,  /* Command in progress */
PQTRANS_INTRANS, /* Idle, within transaction block */
PQTRANS_INERROR, /* Idle, within failed transaction */
PQTRANS_UNKNOWN  /* Cannot determine status */
```

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
    ECPGdebug(1, stderr);

    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb as con1;
    EXEC SQL DROP TABLE IF EXISTS T1;
    int a = ECPGtransactionStatus("con1");
    printf("%d\n", a);
    EXEC SQL CREATE TABLE T1(a int);
    EXEC SQL COMMIT;
    return (0);
}
```

Result (**[44153]** is a randomly generated thread ID):

```
[44153]: ECPGdebug: set to 1
[44153]: ECPGconnect: opening database testdb on <DEFAULT> port <DEFAULT>
[44153]: ecpg_execute on line 12: query: drop table if exists T1; with 0 parameter(s) on connection con1
[44153]: ecpg_execute on line 12: using PQexec
[44153]: ecpg_execute on line 12: OK: DROP TABLE
2
[44153]: ecpg_execute on line 15: query: create table T1 ( a int ); with 0 parameter(s) on connection con1
[44153]: ecpg_execute on line 15: using PQexec
[44153]: ecpg_execute on line 15: OK: CREATE TABLE
[44153]: ECPGtrans on line 16: action "commit"; connection "con1"
```

ECPGfree_auto_mem()

Releases all the memory allocated for output host variables. This function is called (return\exit) when the program ends.

Example:

Prerequisites: The **testdb** database has been created.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
  EXEC SQL BEGIN DECLARE SECTION;
  int *ip1=0;
  char **cp2=0;
  int *ipointer1=0;
  int *ipointer2=0;
  int column;
  EXEC SQL END DECLARE SECTION;
  int i;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  EXEC SQL WHENEVER SQLERROR DO sqlprint();
  EXEC SQL CONNECT TO testdb;

  /* Create the testdb database in advance. */
  EXEC SQL CREATE TABLE test (a int, b text);
  EXEC SQL INSERT INTO test VALUES (1, 'one');
  EXEC SQL INSERT INTO test VALUES (2, 'two');
  EXEC SQL INSERT INTO test VALUES (NULL, 'three');
  EXEC SQL INSERT INTO test VALUES (NULL, NULL);

  EXEC SQL ALLOCATE DESCRIPTOR mydesc;
  EXEC SQL SELECT * INTO SQL DESCRIPTOR mydesc FROM test;
  EXEC SQL GET DESCRIPTOR mydesc :column=COUNT;
  EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :ip1=DATA, :ipointer1=INDICATOR;
  EXEC SQL GET DESCRIPTOR mydesc VALUE 2 :cp2=DATA, :ipointer2=INDICATOR;

  printf("Result (%d columns):\n", column);
  for (i=0;i < sqlca.sqlerrd[2];++i)
  {
    if (ipointer1[i]) printf("NULL, ");
    else printf("%d, ",ip1[i]);

    if (ipointer2[i]) printf("NULL, ");
    else printf("'%s', ",cp2[i]);
    printf("\n");
  }
  ECPGfree_auto_mem();
  printf("\n");

  EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  return 0;
}
```

The result is as follows:

```
Result (2 columns):
1, 'one',
2, 'two',
NULL, 'three',
NULL, NULL,
```

5.10.4.6 SQL Descriptor Areas

5.10.4.6.1 Overview

An SQL descriptor area is a sophisticated method for processing the result of a SELECT, FETCH, or DESCRIBE statement.

An SQL descriptor area groups the data of one row of data together with metadata items into one data structure. ECPG provides two ways to use descriptor areas: the named SQL descriptor areas and the C-structure SQLDAs.

5.10.4.6.2 Named SQL Descriptor Areas

A named SQL descriptor area consists of a header and one or more item descriptor areas. The header contains information concerning the entire descriptor area, whereas each item descriptor area basically describes one column in the result row.

- Before you can use an SQL descriptor area, you need to allocate one.
  ```
  EXEC SQL ALLOCATE DESCRIPTOR identifier;
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When you do not need the descriptor area anymore, you should deallocate it.
  ```
  EXEC SQL DEALLOCATE DESCRIPTOR identifier;
  ```
- To use a descriptor area, declare it using the INTO clause.
  ```
  EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;
  ```
  If the result set is empty, the descriptor area still contains metadata from the query.
- For a prepared query that has not been executed, you can use DESCRIBE to obtain metadata from its result set.
  ```
  EXEC SQL BEGIN DECLARE SECTION;
  char *sql_stmt = "SELECT * FROM table1";
  EXEC SQL END DECLARE SECTION;

  EXEC SQL PREPARE stmt1 FROM :sql_stmt;
  EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
  ```
  In DESCRIBE and FETCH statements, the INTO and USING keywords are used similarly: they produce a result set and metadata in a descriptor area.
- To retrieve the value of a descriptor area from the header and store it into a host variable, use the following command:
  ```
  EXEC SQL GET DESCRIPTOR name :hostvar = field;
  ```
- Currently, there is only one header descriptor area defined: **COUNT**, which tells how many item descriptor areas exist (that is, how many columns are contained in the result set). The host variable needs to be of an integer type. To retrieve a specific value from an item descriptor area, use the following command:
  ```
  EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;
  ```
  num can be a literal integer or a host variable containing an integer. Possible data types are:
  - CARDINALITY (integer): number of rows in the result set.
  - DATA: actual data item (therefore, the data type of this field depends on the query).
  - DATETIME_INTERVAL_CODE (integer): When **TYPE** is **9**, **DATETIME_INTERVAL_CODE** will have a value of **1** for DATE, **2** for TIME, **3** for TIMESTAMP, **4** for TIME WITH TIME ZONE, or **5** for TIMESTAMP WITH TIME ZONE.
  - INDICATOR (integer): indicator (indicating a null value or a value truncation).
  - LENGTH (integer): length of data in characters.
  - NAME (string): column name.
  - OCTET_LENGTH (integer): length (in bytes) of the character representation of data.
  - PRECISION (integer): precision (for the numeric type).
  - RETURNED_LENGTH (integer): length of data in characters.
  - RETURNED_OCTET_LENGTH (integer): length (in bytes) of the character representation of data.
  - SCALE (integer): ratio (for the numeric type).
  - TYPE (integer): numeric code of the data type of the column.
- To retrieve a field value and store it in a host variable, use the following command:
  ```
  EXEC SQL GET DESCRIPTOR mydesc  VALUE num :hostvar = field
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

num can be a literal integer or a host variable containing an integer. Possible fields are:

- DATA
- Actual data item (therefore, the data type of this field depends on the query).
- NAME (string)
- Field name

- To manually create a descriptor area to provide input parameters for a query or cursor, use the following command:
```
EXEC SQL SET DESCRIPTOR name VALUE numfield = :hostvar;
```
- To retrieve multiple rows of records in a FETCH statement and use a host variable of the array type to store data, use the following commands:
```
EXEC SQL BEGIN DECLARE SECTION;
int id[5];
EXEC SQL END DECLARE SECTION;
EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
```

5.10.4.6.3 SQLDA

An SQL descriptor area (SQLDA) is a C language structure used to dynamically process the result set or input parameters of an SQL query. It allows programs to process columns or parameters of unknown quantity and type at runtime.

```
EXEC SQL include sqlda.h;
sqlda_t        *mysqlda;
EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR mysqlda;
```

Note that the SQL keyword is omitted. The paragraphs about the use cases of the INTO and USING keywords in **Named SQL Descriptor Areas** also apply here. In a DESCRIBE statement, the DESCRIPTOR keyword can be completely omitted if the INTO keyword is used.

```
EXEC SQL DESCRIBE prepared_statement INTO mysqlda;
```

SQLDA Data Structures

SQLDAs use three data structures: sqlda_t, sqlvar_t, and struct sqlname.

- sqlda_t structure

  The sqlda_t structure stores the overall information about the result set of a query. Here is its definition:

```
struct sqlda_struct {
    char sqldaid[8];            // It contains the literal string "SQLDA".
    long sqldabc;               // It contains the size of the allocated space in bytes.
    short sqln;                 // It contains the number of input parameters or the number of columns in
the output result set.
    short sqld;                 // It contains the number of columns in a result set.
    struct sqlda_struct *desc_next; // It holds a pointer to the next SQLDA in a linked list.
    struct sqlvar_struct sqlvar[1]; // This is the array of the columns in the result set.
};
typedef struct sqlda_struct sqlda_t;
```

  The meaning of the fields is:

  - **sqldaid**: It contains the literal string "SQLDA".
  - **sqldabc**: It contains the size of the allocated space in bytes.
  - **sqln**: It contains the number of input parameters or the number of columns in the output result set.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **sqld**: It contains the number of columns in a result set.
  - **desc_next**: It holds multiple linked result sets.
  - **sqlvar**: It holds the data and metadata of each column.
- sqlvar_t structure

  The sqlvar_t structure holds a column value and metadata such as type and length. Here is its definition:

```
struct sqlvar_struct {
    short sqltype;              // It contains the type identifier of the column.
    short sqllen;               // It contains the binary length of the column.
    char *sqldata;              // It points to the data in the column.
    short *sqlind;              // It points to the null indicator.
    struct sqlname sqlname;     // It contains the name of the column.
};
typedef struct sqlvar_struct sqlvar_t;
```

  The meaning of the fields is:

  - **sqltype**: It contains the data type of the column (for example, integer or character).
  - **sqllen**: It contains the data length of the column.
  - **sqldata**: It points to the memory area that stores data.
  - **sqlind**: It specifies whether the column is null. **0** means not null, while **-1** means null.
  - **sqlname**: It contains the name of the column.
- struct sqlname structure

  The sqlname structure holds a column name. Here is its definition:

```
#define NAMEDATALEN 64
struct sqlname {
    short length;               // It contains the length of the column name.
    char data[NAMEDATALEN];     // It contains the actual column name.
};
```

  The meaning of the fields is:

  - **length**: It contains the length of the column name.
  - **data**: It contains the actual column name.

Retrieving a Result Set Using an SQLDA

The general steps to retrieve a query result set through an SQLDA are:

Step 1 Declare an sqlda_t structure to receive the result set.

Step 2 Execute the **FETCH**, **EXECUTE**, or **DESCRIBE** command to process a query for which an SQLDA has been declared.

Step 3 Check the number of records in the result set by looking at sqln, a member of the sqlda_t structure.

Step 4 Fetch the values of each column from sqlvar[0], sqlvar[1], and other members of the sqlda_t structure.

Step 5 Go to next row (sqlda_t) by following the desc_next pointer, a member of the sqlda_t structure.

Step 6 Repeat the preceding steps as required.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example:

```
/* Declare an sqlda_t structure to receive the result set. */
sqlda_t *sqlda1;

/* Specify the SQLDA in a command. This is a FETCH command example. */
EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;

/* Run a loop following the linked list to retrieve the rows. */
sqlda_t *cur_sqlda;
for (cur_sqlda = sqlda1;
     cur_sqlda != NULL;
     cur_sqlda = cur_sqlda->desc_next)
{
    ...
}

/* Inside the loop, run another loop to retrieve the data of each column in the row (sqlvar_t structure). */
for (i = 0; i < cur_sqlda->sqld; i++)
{
    sqlvar_t v = cur_sqlda->sqlvar[i];
    char *sqldata = v.sqldata;
    short sqllen  = v.sqllen;
    ...
}

/* To fetch the values of a column, check the sqltype value, a member of the sqlvar_t structure. Then switch
to an appropriate way, depending on the column type, to copy data from the sqlvar field to a host variable.
*/
char var_buf[1024];
switch (v.sqltype)
{
    case ECPGt_char:
        memset(&var_buf, 0, sizeof(var_buf));
        memcpy(&var_buf, sqldata, (sizeof(var_buf) <= sqllen ? sizeof(var_buf) - 1 : sqllen));
        break;

    case ECPGt_int:
        memcpy(&intval, sqldata, sqllen);
        snprintf(var_buf, sizeof(var_buf), "%d", intval);
        break;
    ...
}
```

Passing Query Parameters Using an SQLDA

The general steps to use an SQLDA to pass input parameters to a prepared query are:

Step 1 Create a prepared query (prepared statement).

Step 2 Declare an sqlda_t structure as an SQLDA.

Step 3 Allocate a memory area for the SQLDA.

Step 4 Set (copy) input values in the allocated memory.

Step 5 Open a cursor with specifying the SQLDA.

Step 6 Process the query result.

Step 7 Free up memory.

----End

Example:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
/* First, create a prepared statement. */
   EXEC SQL BEGIN DECLARE SECTION;
   char query[1024] = "SELECT d.oid, * FROM pg_database d, pg_stat_database s WHERE d.oid = s.datid
AND (d.datname = ? OR d.oid = ?)";
   EXEC SQL END DECLARE SECTION;
   EXEC SQL PREPARE stmt1 FROM :query;

   /* Allocate memory for an SQLDA and set the number of input parameters in sqln, a member variable of
the sqlda_t structure.
    * When two or more input parameters are required for the prepared query, the application has to
allocate additional memory space which is calculated by (Number of parameters – 1) × Size of sqlvar_t.
    * The example shown here allocates memory space for two input parameters.
    */
   sqlda_t *sqlda2;
   sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
   memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
   sqlda2->sqln = 2; /* Number of input variables. */

   /* After memory allocation, store the parameter values into the sqlvar[] array. (This is the same array
used for retrieving column values when the SQLDA is receiving a result set.)
    In this example, the input parameters are "testdb" (string type) and 1 (integer type). */
   sqlda2->sqlvar[0].sqltype = ECPGt_char;
   sqlda2->sqlvar[0].sqldata = "testdb";
   sqlda2->sqlvar[0].sqllen  = 8;
   int intval = 1;
   sqlda2->sqlvar[1].sqltype = ECPGt_int;
   sqlda2->sqlvar[1].sqldata = (char *) &intval;
   sqlda2->sqlvar[1].sqllen  = sizeof(intval);

   /* By opening a cursor and specifying the SQLDA that was set up beforehand, the input parameters are
passed to the prepared statement. */
   EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;

   /* Finally, after using input SQLDAs, the allocated memory space must be freed explicitly, unlike SQLDAs
used for receiving query results. */
   free(sqlda2);
```

5.10.4.7 Compatibility Between ECPG and Pro*C

GaussDB employs ECPG, an embedded SQL preprocessor for C programs, while ORA-compatible databases employ the Pro*C preprocessor. ECPG exhibits distinct behavior and semantics from Pro*C in compiling and executing commands, syntax, and embedded statements.

Specific differences between ECPG and Pro*C include:

- ECPG does not support EXEC SQL CONTEXT ALLOCATE, EXEC SQL CONTEXT USE, or EXEC SQL CONTEXT FREE.

  NOTE

  Currently, ECPG does not support context allocation, use, and release. It operates with an independent memory management mechanism. In multi-thread mode, ECPG establishes connections, executes SQL statements, and releases resources separately in each thread. This mirrors the processing logic of Pro*C in multi-thread mode, where each thread also handles context allocation and release.

- ECPG does not support EXEC SQL COMMIT WORK RELEASE.

  NOTE

  In ECPG, the COMMIT statement is executed without the **RELEASE** option. To release resources, you must execute an additional statement like EXEC SQL DISCONNECT or EXEC SQL CLOSE. In contrast, in Pro*C, EXEC SQL COMMIT includes the **RELEASE** option, which can release all resources (such as connections and cursors) held by the program.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- ECPG does not support EXEC SQL ENABLE THREAD.

  NOTE

  To enable macro definition in ECPG's compilation options, you need to define ENABLE_THREAD_SAFETY in the .pgc file of the main function.

- ECPG does not support features or syntaxes such as stored procedures, packages, anonymous blocks, and flashback.

5.10.5 Parameters Related to Log Output

To control log output and better understand the operating status of the database, you can modify specific parameters in the **gaussdb.conf** file in the instance data directory.

Table 5-147 describes the adjustable configuration parameters.

Table 5-147 Configuration parameters

Parameter	Description	Value Range	Remarks
client_min_messages	Level of messages to be sent to the client.	• debug5 • debug4 • debug3 • debug2 • debug1 • log • info • notice • warning • error • fatal • panic Default value: **notice**	All messages at or above the set level will be sent. In general, the lower the set level, the more messages will be sent to the client. For client log messages, if **client_min_messages** is set to **fatal** or **panic**, its effective value will be forcibly adjusted to **error**. Regardless of the level set in **client_min_messages**, messages at the **info** level are always sent to the client.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range	Remarks
log_min_messages	Level of messages to be written into server logs.	• debug5 • debug4 • debug3 • debug2 • debug1 • log • info • notice • warning • error • fatal • panic Default value: **warning**	All messages at or above the set level will be written. In general, the lower the set level, the more messages will be written into server logs. For server log messages, if **log_min_messages** is set to a level not higher than **error**, all messages at the **log** level will be written into server logs.
log_min_error_statement	Level of SQL error statements to be written into server logs.	• debug5 • debug4 • debug3 • debug2 • debug1 • log • info • notice • warning • error • fatal • panic Default value: **error**	All SQL statements corresponding to messages at or above the set level will be written. In general, the lower the set level, the more SQL statements will be written into server logs. For SQL messages in server logs, if the level is set to **error** or lower, the SQL statements corresponding to messages at the **log** level will be written into server logs. Only system administrators can modify this parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range	Remarks
log_min_duration_statement	Minimum execution duration of a statement. If the execution duration of a statement is equal to or longer than the set milliseconds, the statement and its duration will be recorded in logs. Enabling this function can help you track the query attempts to be optimized.	INT type. Default value: **–1**. Unit: millisecond	The value **–1** indicates that the function is disabled. Only a system administrator is allowed to modify this parameter.
log_connections/ log_disconnections	Specifies whether to record a server log message when each session is connected or disconnected.	• **on**: The system records a log server when each session is connected or disconnected. • **off**: The system does not record a log server when each session is connected or disconnected. Default value: **off**	-
log_duration	Specifies whether to record the duration of each executed statement.	• **on**: The system records the duration of each executed statement. • **off**: The system does not record the duration of each executed statement. Default value: **off**	Only a system administrator is allowed to modify this parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range	Remarks
log_statement	SQL statements to be recorded in logs.	• **none**: The system does not record any SQL statements. • **ddl**: The system records data definition statements. • **mod**: The system records data definition statements and data operation statements. • **all**: The system records all statements. Default value: **none**	Only a system administrator is allowed to modify this parameter.
log_hostname	Specifies whether to record host names.	• **on**: The system records host names. • **off**: The system does not record host names. Default value: **off**	By default, connection logs only record the IP addresses of connected hosts. With this function, the host names will also be recorded. This parameter affects the display of audit results, the PG_STAT_ACTIVITY system view, and the GUC parameter **log_line_prefix**.

Table 5-148 describes the log level parameters.

Table 5-148 Description of log level parameters

Level	Description
DEBUG[1-5]	Provides information that can be used by developers. Level 1 is the lowest level whereas level 5 is the highest level.
INFO	Provides information about users' hidden requests, for example, information about the VACUUM VERBOSE process.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Level	Description
NOTICE	Provides information that may be important to users, for example, truncations of long identifiers or indexes created as a part of a primary key.
WARNING	Provides warning information for users, for example, COMMIT out of transaction blocks.
ERROR	Reports an error that causes a command to terminate.
LOG	Reports information that administrators may be interested in, for example, the activity levels of checkpoints.
FATAL	Reports the reason that causes a session to terminate.
PANIC	Reports the reason that causes all sessions to terminate.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6 User-Defined Functions

When the cluster is started, the UDF master process is started in addition to CNs and DNs. To execute a UDF in fenced mode, the UDF master process forks itself to UDF worker processes, and UDF worker processes execute the UDF in fenced mode.

6.1 PL/SQL Functions

PL/SQL is a loadable procedural language.

Functions created using PL/SQL can be used in any place where you can use built-in functions. For example, you can use calculation functions, definition operators, and index expressions to create complex conditions.

SQL is used by most databases as a query language. It is portable and easy to learn. Each SQL statement must be executed independently by a database server.

This means that the client application performs the following processes for each query: send a query to the database server, wait for the query to be received, receive and process the result, perform related calculation, and then send more queries to the server. If the client and the database server are not on the same machine, this process also causes inter-process communication and network load.

PL/SQL enables a whole computing part and a series of queries to be grouped inside a database server. This makes procedural language available and SQL easier to use. In addition, the client/server communication cost is reduced.

- Extra round-trip communication between clients and servers is eliminated.
- Intermediate results that are not required by clients do not need to be sorted or transmitted between the clients and servers.
- Parsing can be skipped in multiple rounds of queries.

PL/SQL can use all data types, operators, and functions in SQL. To create functions in PL/SQL, use the CREATE FUNCTION syntax.

PL/SQL is a loadable procedural language that operates similarly to stored procedures. A key difference is that stored procedures do not return values, whereas PL/SQL functions do.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about the syntax and stored procedures in non-multi-primary mode, see "SQL Reference > SQL Syntax" and "Stored Procedure" in *Reference*.

In non-multi-primary mode, user-defined functions accept XML data as input parameters, output parameters, user-defined variables, and return values of user-defined functions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7 Feature Usage Guidelines

7.1 Autonomous Transaction

An autonomous transaction is an independent transaction that is started during the execution of a primary transaction. Committing and rolling back an autonomous transaction does not affect the data that has been committed by the primary transaction. In addition, an autonomous transaction is not affected by the primary transaction.

Autonomous transactions are defined in stored procedures, functions, and anonymous blocks, and are declared using the **PRAGMA AUTONOMOUS_TRANSACTION** keyword.

7.1.1 Specifications and Limitations

- In non-multi-primary mode, executing an autonomous transaction starts an autonomous transaction session in the background. You can set **max_concurrent_autonomous_transactions** to limit the maximum number of concurrent autonomous transactions (value range: 0 to 10000). If **max_concurrent_autonomous_transactions** is set to **0**, autonomous transactions cannot be executed.
- After a new autonomous transaction session is started in non-multi-primary mode, it uses the default **session** parameter and does not share objects (including session-level variables, local temporary variables, and global temporary table data) from the main transaction session.
- After a new autonomous transaction session is started in non-multi-primary mode, it synchronizes the following GUC parameters from the main transaction session: **session_max_dynamic_memory**, **behavior_compat_options**, and **nls_date_format**. For details about these parameters, see "Database GUC Parameters > GUC Parameters" in *Reference*.
- While the theoretical upper limit for autonomous transactions in non-multi-primary mode is 10,000, the actual limit is dynamic. For details, see the description of the GUC parameter **max_concurrent_autonomous_transactions**.
- In non-multi-primary mode, autonomous transactions are affected by the communication buffer. The size of information returned to the client is limited

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

by the length of the communication buffer. If the returned information exceeds the length of the communication buffer, an error will be raised.

- In non-multi-primary mode, during the execution of an autonomous transaction, the main transaction places a lock on it. If this lock becomes residual due to exceptions, it must be cleared using gs_clean. The clearance period is controlled by the GUC parameter **gs_clean_timeout**, which defaults to 60 seconds. Autonomous transaction locks are not subject to the lock timeout limit of 2,147,483 seconds. However, if the execution time of an autonomous transaction exceeds this limit, a lock timeout error will be raised.
- In non-multi-primary mode, autonomous transactions enforce a 5-second connection timeout interval and allow up to five connection attempts. During connection establishment, they do not immediately respond to signals, but perform a signal check before each connection attempt. Timeout errors may occur in high concurrency, high CPU usage, high memory usage, and thread pool scale-out scenarios.
- If the PRAGMA AUTONOMOUS_TRANSACTION syntax is declared in PACKAGE SPECIFICATION or PACKAGE BODY SPECIFICATION, a package can be created, but the autonomous transaction does not take effect.
- A trigger function does not support autonomous transactions.

```
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 int, id2 int, id3 int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(new.id1, new.id2, new.id3);
RETURN new;END$$ LANGUAGE plpgsql;
ERROR:  Triggers do not support autonomous transactions
DETAIL:  N/A

gaussdb=# DROP TABLE test_trigger_des_tbl;
DROP TABLE
```

- Autonomous transactions cannot be called by non-top-layer anonymous blocks (but can only be called by top-layer autonomous transactions, including stored procedures, functions, and anonymous blocks).

```
gaussdb=# CREATE TABLE t1(a INT ,b TEXT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# DECLARE
 --PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
  DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
  INSERT INTO t1 VALUES(1,'CAN YOU ROLLBACK!');
 END;
 INSERT INTO t1 VALUES(2,'I WILL ROLLBACK!');
 ROLLBACK;
END;
/
JUST USE CALL.
ANONYMOUS BLOCK EXECUTE

gaussdb=# SELECT * FROM t1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 a | b
---+---
(0 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

- In an autonomous transaction, the **ref cursor** parameter can be passed only through an OUT parameter of a stored procedure—not through an IN/INOUT parameter or a function. In non-multi-primary mode, the **ref cursor** parameter can only be passed through an OUT parameter of an autonomous transaction within a stored procedure and cannot be directly called from a client, such as gsql or JDBC.

```
-- Create a table.
gaussdb=# CREATE TABLE sections(section_id INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'section_id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
```

  a. Pass the **ref cursor** parameter through an OUT parameter of a stored procedure (supported).

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(OUT c1 REFCURSOR)
IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;

END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
  temp NUMBER(4);
BEGIN
  proc_sys_ref(c1);
  IF c1%ISOPEN THEN
    RAISE NOTICE '%','OK';
 END IF;

  LOOP
    FETCH c1 INTO temp;
    RAISE NOTICE '%',c1%ROWCOUNT;
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_sys_call();
NOTICE:  OK
NOTICE:  1
NOTICE:  2
NOTICE:  3
NOTICE:  4
NOTICE:  4
 proc_sys_call
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
---------------

(1 row)
```

b. Pass the **ref cursor** parameter through an IN or INOUT parameter of a stored procedure (not supported).

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(IN c1 REFCURSOR)
IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  temp NUMBER(4);
BEGIN
  IF c1%ISOPEN THEN
    RAISE NOTICE '%','OK';
  END IF;

  LOOP
    FETCH c1 INTO temp;
    RAISE NOTICE '%',c1%ROWCOUNT;
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
  temp NUMBER(4);
BEGIN
  OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
  proc_sys_ref(c1);
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_sys_call();
ERROR:  Unsupported: ref_cursor parameter is not supported for autonomous transactions.
CONTEXT:  SQL statement "CALL proc_sys_ref(c1)"
PL/pgSQL function proc_sys_call() line 7 at PERFORM
```

c. Pass the **ref cursor** parameter through the return of a function (not supported).

```
gaussdb=# DROP PROCEDURE IF EXISTS proc_sys_ref;
gaussdb=# CREATE OR REPLACE function proc_sys_ref() RETURN SYS_REFCURSOR IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
        c1 SYS_REFCURSOR;
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
 return c1;
END;
/
ERROR:  Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL:  N/A
```

d. Pass the **ref cursor** parameter through an OUT parameter of a function (not supported).

```
gaussdb=# CREATE OR REPLACE FUNCTION proc_sys_ref(c1 OUT SYS_REFCURSOR)
RETURN SYS_REFCURSOR
IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
 RETURN 1;
END;
/
ERROR:  Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL:  N/A
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

e. In non-multi-primary mode, directly call an autonomous transaction procedure that contains the output parameter **ref cursor** from a client, such as gsql or JDBC. (This is not supported, and cursor data cannot be read in this case.)

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(OUT c1 REFCURSOR)
IS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
END;
/
CREATE PROCEDURE
gaussdb=# begin;
BEGIN
gaussdb=# call proc_sys_ref(null);
        c1
--------------------
 <unnamed portal 1>
(1 row)

gaussdb=# fetch "<unnamed portal 1>";
ERROR:  cursor "<unnamed portal 1>" does not exist
gaussdb=# end;
ROLLBACK
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
-- Drop the table.
gaussdb=# DROP TABLE sections;
DROP TABLE
```

- In non-multi-primary mode, autonomous transactions (IMMUTABLE or STABLE type) cannot be pushed down.

```
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_test_in_p_116(num1 INT )
IMMUTABLE
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
PERFORM pg_sleep(1);
END;
/
ERROR:  Autonomous transactions do not support STABLE/IMMUTABLE.
DETAIL:  Please remove stable/immutable.

gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_test_in_p_117(num1 INT ) STABLE AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
PERFORM pg_sleep(1);
END;
/
ERROR:  Autonomous transactions do not support STABLE/IMMUTABLE.
DETAIL:  Please remove stable/immutable.
```

- In non-multi-primary mode, detection is not supported.

```
gaussdb=# CREATE TABLE test_lock (id INT,a DATE);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO test_lock VALUES (10,SYSDATE),(11,SYSDATE),(12,SYSDATE);
INSERT 0 3
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_test_lock(num1 INT,num2 INT) RETURNS
INTEGER LANGUAGE plpgsql AS $$
DECLARE num3 INT := 4;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
UPDATE test_lock SET a=SYSDATE WHERE id =11;
RETURN num1+num2+num3;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
END;
$$;
CREATE FUNCTION

gaussdb=# START TRANSACTION;
START TRANSACTION
gaussdb=# UPDATE test_lock SET a=SYSDATE WHERE id =11;
UPDATE 1
gaussdb=# CALL autonomous_test_lock(1,1);
ERROR:  ERROR:  Lock wait timeout: thread 139874535470848 on node datanode1 waiting for
ShareLock on transaction 16214 after 120000.124 ms
DETAIL:  blocked by hold lock thread 139874577413888, statement <UPDATE test_lock SET
a=SYSDATE WHERE id =11;>, hold lockmode ExclusiveLock.
CONTEXT:  SQL statement "UPDATE test_lock SET a=SYSDATE WHERE id =11"
PL/SQL function autonomous_test_lock(integer,integer) line 5 at SQL statement
referenced column: autonomous_test_lock

gaussdb=# END;
ROLLBACK
gaussdb=# DROP TABLE test_lock;
DROP TABLE
```

- Autonomous transaction functions cannot directly return the record type, nor can they return OUT parameters and the record type simultaneously.

```
gaussdb=# CREATE OR REPLACE FUNCTION auto_func() RETURN RECORD
AS
DECLARE
  TYPE rec_type IS RECORD(c1 INT, c2 INT);
  r rec_type;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  r.c1:=101;
  r.c2:=201;
  RETURN r;
END;
/
CREATE FUNCTION

gaussdb=# SELECT auto_func();
ERROR:  unrecognized return type for PLSQL function.
CONTEXT:  referenced column: auto_func
```

- The isolation level of an autonomous transaction cannot be changed.

```
gaussdb=# CREATE OR REPLACE PROCEDURE auto_func(r INT)
AS
DECLARE
  a INT;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
 a:=r;
END;
/
CREATE FUNCTION
gaussdb=# call auto_func(1);
ERROR:  ERROR:  SET TRANSACTION ISOLATION LEVEL must be called before any query
CONTEXT:  SQL statement "SET TRANSACTION ISOLATION LEVEL SERIALIZABLE"
PL/SQL function auto_func(integer) line 6 at SQL statement
referenced column: auto_func
```

- Autonomous transactions cannot return the setof type.

```
gaussdb=# CREATE OR REPLACE FUNCTION test_set() RETURN SETOF INT
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    RETURN QUERY (SELECT unnest(ARRAY[ARRAY[1, 2], ARRAY[3, 4]]));
END;
/
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ERROR:  Autonomous transactions do not support RETURN SETOF.
DETAIL:  N/A
```

7.1.2 Autonomous Transactions Supported in Stored Procedures

Autonomous transactions can be defined in stored procedures, with the PRAGMA AUTONOMOUS_TRANSACTION identifier. The rest of the syntax follows the standard syntax for creating stored procedures. For more details, see "SQL Reference > SQL Syntax > C > CREATE PROCEDURE" in *Reference*. Below is an example:

```
-- Create a table.
gaussdb=# CREATE TABLE t2(a INT, b INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(1,2);
INSERT 0 1
gaussdb=# SELECT * FROM t2;
 a | b
---+---
 1 | 2
(1 row)

-- Create a stored procedure that contains an autonomous transaction.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_4(a INT, b INT)  AS
DECLARE
    num3 INT := a;
    num4 INT := b;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(num3, num4);
    DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
END;
/
CREATE PROCEDURE

-- Create a common stored procedure that calls an autonomous transaction stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a INT, b INT)  AS
DECLARE
BEGIN
    DBE_OUTPUT.PRINT_LINE('JUST NO USE CALL.');
    INSERT INTO t2 VALUES(666, 666);
    autonomous_4(a,b);
    ROLLBACK;
END;
/
CREATE PROCEDURE

-- Call a common stored procedure.
gaussdb=# SELECT autonomous_5(11,22);
JUST NO USE CALL.
JUST USE CALL.
 autonomous_5
--------------

(1 row)

-- View the table result.
gaussdb=# SELECT * FROM t2 ORDER BY a;
 a  | b
----+----
  1 |  2
 11 | 22
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# DROP TABLE t2;
DROP TABLE
```

In the preceding example, a stored procedure containing an autonomous transaction is finally executed in a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

7.1.3 Autonomous Transactions Supported in Anonymous Blocks

Autonomous transactions can be defined in anonymous blocks, with the PRAGMA AUTONOMOUS_TRANSACTION identifier. The rest of the syntax follows the standard syntax for creating anonymous blocks. For more details, see "Stored Procedure > Declaration Syntax > Anonymous Block" in *Reference*. Below is an example:

```
gaussdb=# CREATE TABLE t1(a INT ,B TEXT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# START TRANSACTION;
START TRANSACTION
gaussdb=# DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
    INSERT INTO t1 VALUES(1,'YOU ARE SO CUTE,WILL COMMIT!');
END;
/
JUST USE CALL.
ANONYMOUS BLOCK EXECUTE

gaussdb=# INSERT INTO t1 VALUES(1,'YOU WILL ROLLBACK!');
INSERT 0 1
gaussdb=# ROLLBACK;
ROLLBACK
gaussdb=# SELECT * FROM t1;
 a |             b
---+-------------------------------
 1 | YOU ARE SO CUTE,WILL COMMIT!
(1 row)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

In the preceding example, an anonymous block containing an autonomous transaction is finally executed before a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

7.1.4 Autonomous Transactions Supported in Functions

Autonomous transactions can be defined in functions, with the PRAGMA AUTONOMOUS_TRANSACTION identifier. The rest of the syntax follows the standard syntax for creating functions. For more details, see "SQL Reference > SQL Syntax > C > CREATE FUNCTION" in *Reference*. Below is an example:

```
gaussdb=# CREATE TABLE t4(a INT, b INT, c TEXT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

gaussdb=# CREATE OR REPLACE FUNCTION autonomous_32(a INT ,b INT ,c TEXT) RETURN INT AS
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO t4 VALUES(a, b, c);
   RETURN 1;
END;
/
CREATE FUNCTION
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_33(num1 INT) RETURN INT AS
DECLARE
   num3 INT := 220;
   tmp INT;
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   num3 := num3/num1;
   RETURN num3;
EXCEPTION
   WHEN DIVISION_BY_ZERO THEN
      SELECT autonomous_32(num3, num1, SQLERRM) INTO tmp;
      ROLLBACK;
      RETURN 0;
END;
/
CREATE FUNCTION
gaussdb=# SELECT autonomous_33(0);
 autonomous_33
---------------
             0
(1 row)

gaussdb=# SELECT * FROM t4;
  a  | b |        c
-----+---+------------------
 220 | 0 | division by zero
(1 row)

gaussdb=# DROP TABLE t4;
DROP TABLE
```

In the preceding example, a function containing an autonomous transaction is finally executed in a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

7.1.5 Package Supporting Autonomous Transaction

Autonomous transactions can be defined in stored procedures or functions within packages, with the PRAGMA AUTONOMOUS_TRANSACTION identifier. The rest of the syntax follows the standard syntax for creating stored procedures or functions within packages. For more details, see "SQL Reference > SQL Syntax > C > CREATE PACKAGE" in *Reference*. Below is an example:

```
gaussdb=# create database test DBCOMPATIBILITY = 'ORA';
CREATE DATABASE
test=# \c test
-- Create a table.
test=# CREATE TABLE t2(a INT, b INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
test=# INSERT INTO t2 VALUES(1,2);
INSERT 0 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
test=# SELECT * FROM t2;
 a | b
---+---
 1 | 2
(1 row)

-- Create a stored procedure or function in a package that contains autonomous transactions.
test=# CREATE OR REPLACE PACKAGE autonomous_pkg AS
  PROCEDURE autonomous_4(a INT, b INT);
  FUNCTION autonomous_32(a INT ,b INT) RETURN INT;
END autonomous_pkg;
/
CREATE PACKAGE
test=# CREATE OR REPLACE PACKAGE BODY autonomous_pkg AS
PROCEDURE autonomous_4(a INT, b INT)  AS
DECLARE
 num3 INT := a;
 num4 INT := b;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(num3, num4);
END;
FUNCTION autonomous_32(a INT ,b INT) RETURN INT AS
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(a, b);
    RETURN 1;
END;
END autonomous_pkg;
/
CREATE PACKAGE BODY
-- Create a common stored procedure that calls a stored procedure or function from a package that
contains autonomous transactions.
test=# CREATE OR REPLACE PROCEDURE autonomous_5(a INT, b INT)  AS
DECLARE
va INT;
BEGIN
    INSERT INTO t2 VALUES(666, 666);
    autonomous_pkg.autonomous_4(a,b);
        va := autonomous_pkg.autonomous_32(a + 1, b + 1);
    ROLLBACK;
END;
/
CREATE PROCEDURE
-- Call a common stored procedure.
test=# SELECT autonomous_5(11,22);
 autonomous_5
--------------

(1 row)

-- View the table result.
test=# SELECT * FROM t2 ORDER BY a;
 a  | b
----+----
  1 |  2
 11 | 22
 12 | 23
(3 rows)

test=# DROP TABLE t2;
DROP TABLE
```

In the preceding example, a stored procedure or function in a package containing autonomous transactions is finally executed in a transaction block that is rolled back, which directly illustrates a characteristic of the autonomous transaction, that

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

is, rollback of the primary transaction does not affect content that has been committed by an autonomous transaction.

7.2 Materialized View

A materialized view is a special physical table compared to an ordinary view. An ordinary view is a virtual table with many application limitations. Any query on a view is actually converted into a query on an SQL statement, and performance is not actually improved. The materialized view serves as a cache to store the results of executed SQL statements. Common operations on materialized views include creating, querying, deleting, and refreshing the views.

Materialized views fall into two types—complete-refresh and fast-refresh—based on their creation rules. Complete-refresh materialized views support only full refreshes, while fast-refresh materialized views support both full and incremental refreshes. Full refreshes will insert all data from the base table to the materialized view, while incremental refreshes insert only the incremental data generated between refresh intervals.

Currently, Ustore does not support creating or using materialized views.

7.2.1 Complete-Refresh Materialized View

7.2.1.1 Overview

Complete-refresh materialized views allow only full refreshes and do not support incremental refreshes.

The syntax for creating a complete-refresh materialized view is similar to CREATE TABLE AS. For details, see "SQL Reference > SQL Syntax > CREATE TABLE AS" in *Reference*. During the creation of a complete-refresh materialized view, you cannot specify a node group, and limitations in GTM-free mode also apply.

7.2.1.2 Support and Limitations

Supported Scenarios

- Generally, the query scope supported by complete-refresh materialized views is the same as that supported by the CREATE TABLE AS statement.
- A distribution key can be specified during the creation of a complete-refresh materialized view.
- Indexes can be created in a complete-refresh materialized view.
- ANALYZE and EXPLAIN are supported.

Unsupported Scenarios

- Complete-refresh materialized views do not support node groups.
- Materialized views do not support INSERT, DELETE, or UPDATE statements. They support only the SELECT statement.
- Ustore does not support creating or using complete-refresh materialized views.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Limitations

- The base table used to create a complete-refresh materialized view must be defined on all DNs, and the node group to which the base table belongs must be an installation group.
- Refreshing or dropping a complete-refresh materialized view places a level-7 lock (which blocks DML and DDL operations) on the base table. If the definition of the materialized view involves multiple tables, carefully design the service logic to prevent deadlocks.

7.2.1.3 Usage

Syntax

- Create a complete-refresh materialized view.
  ```
  CREATE MATERIALIZED VIEW view_name AS query;
  ```
- Refresh a complete-refresh materialized view.
  ```
  REFRESH MATERIALIZED VIEW view_name;
  ```
- Drop a materialized view.
  ```
  DROP MATERIALIZED VIEW view_name;
  ```
- Query a materialized view.
  ```
  SELECT * FROM view_name;
  ```

Parameters

- **view_name**

  Specifies the name of the materialized view to be created.

  Value range: character strings complying with identifier naming conventions.

- **AS query**

  Specifies a **SELECT VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT** or **VALUES** query.

Examples

```
-- Change the default type of a table.
gaussdb=# set enable_default_ustore_table=off;

-- Prepare data.
CREATE TABLE t1(c1 int, c2 int);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES(2, 2);

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

-- Query results from the materialized view.
gaussdb=# SELECT * FROM mv;
 count
-------
     2
(1 row)

-- Insert data into the base table of the materialized view.
gaussdb=# INSERT INTO t1 VALUES(3, 3);

-- Perform a full refresh on the complete-refresh materialized view.
gaussdb=# REFRESH MATERIALIZED VIEW mv;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
REFRESH MATERIALIZED VIEW

-- Query results from the materialized view.
gaussdb=# SELECT * FROM mv;
 count
-------
     3
(1 row)

-- Drop the materialized view and table.
gaussdb=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
gaussdb=# DROP TABLE t1;
DROP TABLE
```

7.2.2 Fast-Refresh Materialized View

7.2.2.1 Overview

Fast-refresh materialized views support incremental refreshes. To do this, you need to manually execute statements that incrementally refresh data from a specified time period within the views.

Unlike complete-refresh materialized views, fast-refresh materialized views apply only to a limited number of scenarios. Currently, materialized views can only be created using the base table scanning statement or the UNION ALL statement.

7.2.2.2 Support and Limitations

Supported Scenarios

- Execute statements to query a single table.
- Perform UNION ALL queries across multiple individual tables.
- Create indexes on materialized views.
- Perform ANALYZE operations on materialized views.
- Create a fast-refresh materialized view based on a node group inherited from base tables (ensure that the base tables reside in the same node group).

Unsupported Scenarios

- Materialized views do not support the Stream, multi-table join, or subquery plans.
- The WITH, GROUP BY, ORDER BY, LIMIT, and WINDOW clauses are unsupported. The DISTINCT and AGG operators are also unsupported. Additionally, subqueries other than UNION ALL are not supported.
- Except for a few ALTER operations, most DDL operations cannot be performed on base tables in materialized views.
- When creating a materialized view, you cannot specify a distribution key for it.
- Materialized views do not support INSERT, DELETE, or UPDATE statements. They support only the SELECT statement.
- Materialized views cannot be created using a temporary, hash bucket, unlogged, or partitioned table. Only the hash distributed table is supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Materialized views cannot be nested (that is, you cannot create one materialized view inside another).
- Materialized views of the UNLOGGED type are not supported, and the WITH syntax is not supported.
- Ustore does not support creating or using fast-refresh materialized views.

Limitations

- Materialized views defined with UNION ALL require each subquery to reference distinct base tables, and the distribution keys of these base tables must be identical. The distribution key of the materialized view is automatically inferred to match that of the base tables.
- The columns defined in a materialized view must include all distribution keys in its base tables.
- Creating, fully refreshing, or dropping a fast-refresh materialized view places a level-7 lock (which blocks DML and DDL operations) on its base tables. If the materialized view is defined as UNION ALL, carefully design the service logic to prevent deadlocks.

7.2.2.3 Usage

Syntax

- Create a fast-refresh materialized view.
  ```
  CREATE INCREMENTAL MATERIALIZED VIEW view_name AS query;
  ```
- Perform a full refresh on a materialized view.
  ```
  REFRESH MATERIALIZED VIEW view_name;
  ```
- Perform an incremental refresh on a materialized view.
  ```
  REFRESH INCREMENTAL MATERIALIZED VIEW view_name;
  ```
- Drop a materialized view.
  ```
  DROP MATERIALIZED VIEW view_name;
  ```
- Query a materialized view.
  ```
  SELECT * FROM view_name;
  ```

Parameters

- **view_name**

  Specifies the name of the materialized view to be created.

  Value range: character strings complying with identifier naming conventions.

- **AS query**

  Specifies a **SELECT VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT** or **VALUES** query.

Examples

```
-- Change the default type of a table.
gaussdb=# SET enable_default_ustore_table=off;

-- Prepare data.
CREATE TABLE t1(c1 int, c2 int);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES(2, 2);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM t1;
CREATE MATERIALIZED VIEW

-- Insert data.
gaussdb=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

-- Perform an incremental refresh on the materialized view.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query results from the materialized view.
gaussdb=# SELECT * FROM mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

-- Insert data.
gaussdb=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

-- Perform a full refresh on the materialized view.
gaussdb=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query results from the materialized view.
gaussdb=# select * from mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
(4 rows)

-- Drop the materialized view and table.
gaussdb=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
gaussdb=# DROP TABLE t1;
DROP TABLE
```

7.3 Encrypted Equality Query

7.3.1 Overview

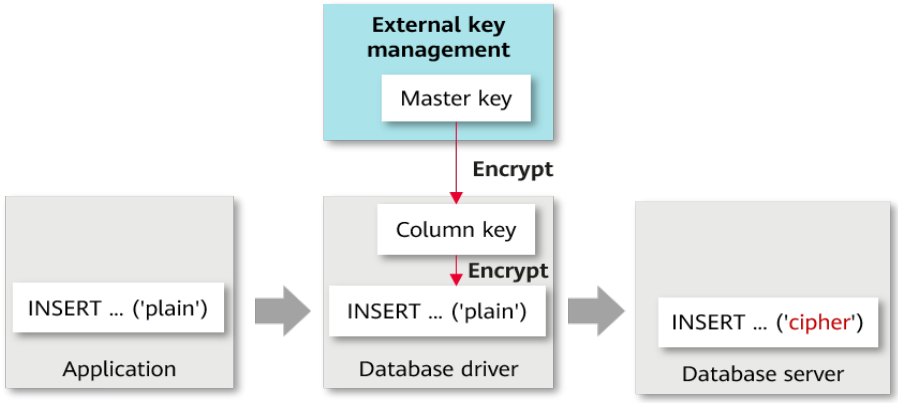
As enterprises migrate data to the cloud, they face growing challenges in ensuring data security and privacy. Encrypted databases address these privacy concerns throughout the entire data lifecycle, including network transmission, data storage, and data operation status. Furthermore, encrypted databases enforce data privacy permission separation in cloud environments by separating data read permissions between data owners and data administrators. The encrypted equality query is used to solve equality query issues of ciphertext data.

Encryption Model

A fully-encrypted database uses a multi-level encryption model comprising three core objects: data, column key, and master key. Their relationships are illustrated in **Figure 7-1**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-1 Encryption model



Below is an introduction to the three objects:

- **Data**, including:
  a. Data contained in SQL syntax, such as 'data' in INSERT... VALUES ('data')
  b. Query results returned from the database server, such as the results of the SELECT syntax

  NOTE

  An encrypted database encrypts data of encrypted columns in the SQL syntax in the driver and decrypts the query result of the encrypted columns returned from the database server.
- **Column key**: Data is encrypted by using column keys. The column keys are generated by the database driver or manually imported by users. The column key ciphertext is stored on the database server.
- **Master key**: Column keys are encrypted by using master keys. The master keys are generated and stored by an external key manager. The database driver automatically accesses the external key manager to encrypt and decrypt column keys.

Overall Process

Using a fully-encrypted database involves the following four phases. This section describes the overall process, with detailed instructions provided in **Using gsql to Operate on an Encrypted Database** and **Using JDBC to Operate on an Encrypted Database**.

1. Master key generation phase: First, generate a master key in Huawei Cloud key management service (KMS). Then prepare parameters for the database to access the master key.

2. DDL statement execution phase: In this phase, you can use the key syntax of the encrypted database to define the master key and column key in sequence, define the table, and specify a column in the table as an encrypted column. When defining the master key and column key, you need to access the master key generated in the previous phase.

3. DML statement execution phase: After an encrypted table is created, you can directly execute syntax including but not limited to INSERT, SELECT, UPDATE, and

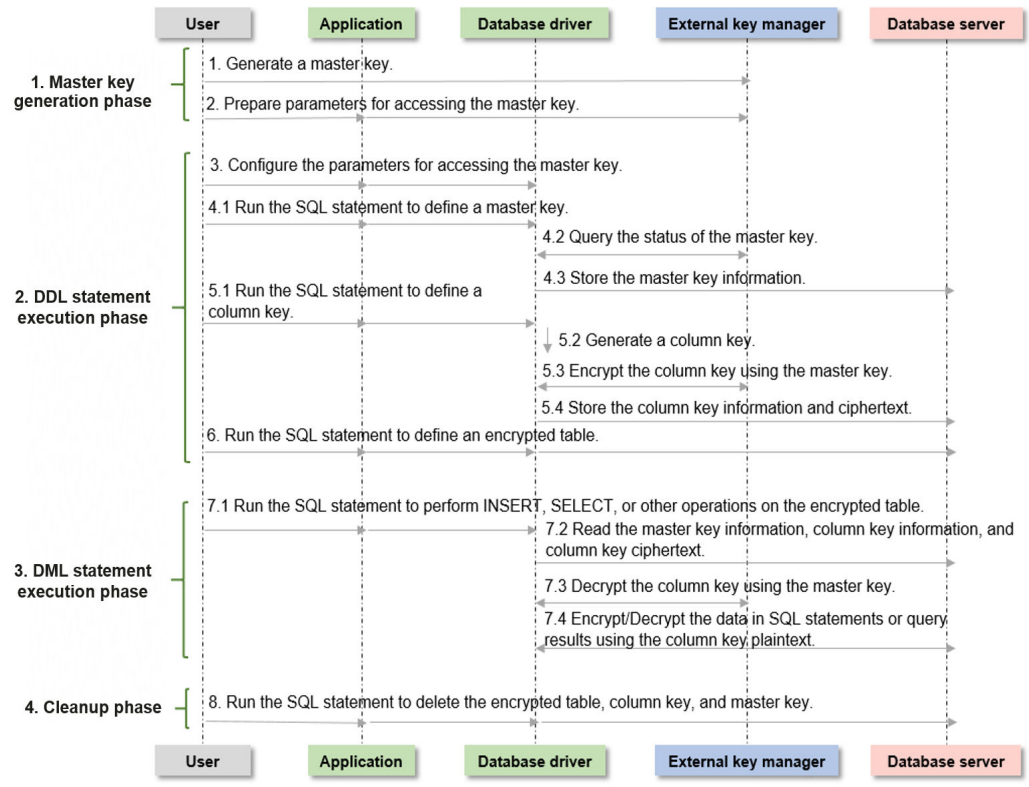
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DELETE. The database driver automatically encrypts and decrypts data of the encrypted column based on the encryption definition in the previous phase.

4. Cleanup phase: Delete the encrypted table, column key, and master key in sequence.

Figure 7-2 shows the overall process.

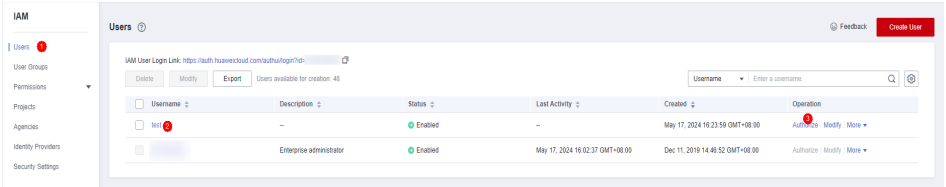
Figure 7-2 Overall process



Master Key Generation Phase

When you use an encrypted database for the first time, you need to use an external key manager to generate at least one master key. The operation procedure is as follows:

- Huawei Cloud scenario
  a. Access the Huawei Cloud website, create an account, and log in with the account.
  b. Search for and access the IAM service. On the **Users** page, click **Create User** to create an IAM user, set a password, and grant the DEW permission to it.

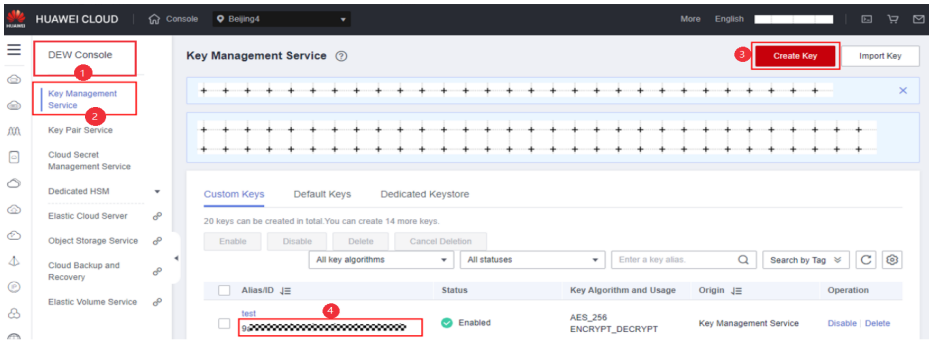


 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

c. Return to the login page and choose to log in as the IAM user created in the previous step. The subsequent operations are all performed by the IAM user.

d. Choose **Key Management Service** and click **Create Key** to create at least one key (master key).

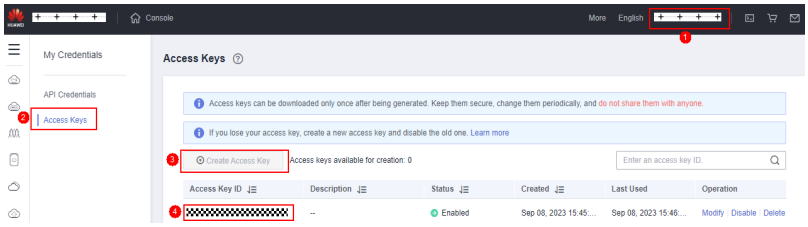
e. Remember the master key ID. Each master key has a key ID. When using encrypted data, you need to configure the master key ID. The database driver accesses the master key through the RESTful API.



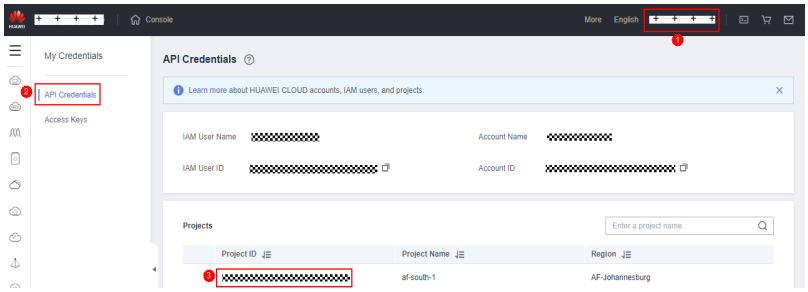
After the master key is generated, you need to prepare parameters for the data driver to access the master key, such as the IAM username and project ID. Huawei Cloud offers two identity authentication modes. These modes differ in the number and types of parameters required. You can choose either mode as required. To obtain these parameters, perform the following steps:

- **Mode 1**: AK/SK authentication

  a. AK and SK: Log in to the Huawei Cloud console, click the username in the upper right corner, and choose **My Credentials**. On the displayed page, click **Access Keys**. Click **Create Access Key** to create an AK/SK pair. After the creation is successful, you can download the AK and SK.



  b. Project ID: On the Huawei Cloud console, click the username in the upper right corner, and choose **My Credentials**. On the displayed page, click **API Credentials** to find the project ID.

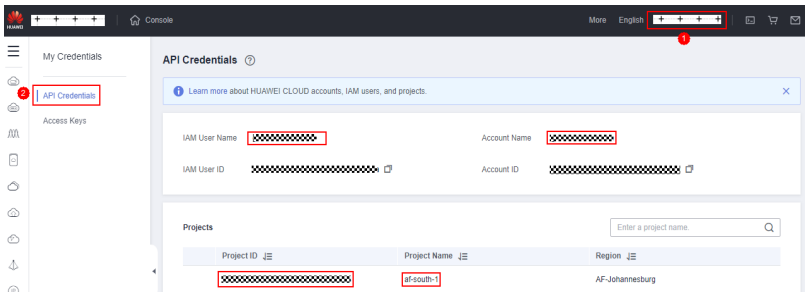


 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

c. KMS server address: https://kms.*Project*.myhuaweicloud.com/v1.0/*Project ID*/kms

- **Mode 2**: Account/Password authentication

a. IAM username, account name, project, and project ID: On the Huawei Cloud console, click the username in the upper right corner and choose **My Credentials**. On the displayed page, you can obtain the IAM username, account name, project, and project ID, as shown in the following figure.



b. IAM server address: https://iam.*Project*.myhuaweicloud.com/v3/auth/tokens

c. IAM user password: password for the IAM user

d. KMS server address: https://kms.*Project*.myhuaweicloud.com/v1.0/*Project ID*/kms

7.3.2 Using gsql to Operate on an Encrypted Database

Using gsql to Execute SQL Statements

Generate master keys and confirm their access parameters before executing the SQL statements in this section.

This section uses a complete execution process as an example to describe how to use the encrypted database syntax, including three phases: DDL statement execution, DML statement execution, and cleanup.

Step 1 Connect to the database and use the **-C** parameter to enable the full encryption function.

```
gsql -p PORT -d DATABASE -h HOST -U USER -W PASSWORD -r -C
```

Step 2 Set parameters for accessing the master key using a meta-command.

Note: There must be no line feed or space in the string starting from **keyType**. Otherwise, the gsql tool cannot identify the entire parameter.

Key management based on the SanSec SJJ1994 HSM:

```
gaussdb=# \key_info keyType=sdf_kms,library_path={Path of the dynamic library},library_version=SJJ1994,kek_algo=SGD_SM4_CBC,mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Before using the SanSec SJJ1994 HSM, you must correctly configure the .ini file and ensure that the HSM can be properly connected.
- **library_path** can be an absolute path or a relative path (relative to *$GAUSSHOME*/**lib**).
- **kek_algo** refers to the algorithm that key encryption keys (KEKs) use to encrypt session keys. Supported algorithms include: SGD_SM4_ECB (default), SGD_SM4_CBC, SGD_SM4_CTR, SGD_AES_ECB, SGD_AES_CBC, and SGD_AES_CTR.
- **dkey_algo** refers to the algorithm that session keys use to encrypt data keys. Supported algorithms include: SGD_SM4_CBC, SGD_SM4_CTR, SGD_AES_CBC, and SGD_AES_CTR.
- **mac_algo** refers to the algorithm that session keys use to verify the integrity of data keys. Supported algorithms include: SGD_SM4_MAC and SGD_AES_MAC.
- **library_version** (optional) is used for customized interconnection. Currently, it can only be set to **SJJ1994**.
- **kek_algo** must match the **KEY_PATH** specified when the CMK is created. That is, both of them must be AES or SM4 series algorithms.
- Currently, the gsql and JDBC drivers are supported.

Kunpeng TEE intrinsic key management:

```
gaussdb=# \key_info keyType=sdf_kms,library_path={Path of the dynamic library},mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC
```

Example: \key_info KeyType=sdf_kms,library_path=./libsdf.so,mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC

Huawei Cloud offers two authentication modes. These modes differ in the number and types of parameters required. You can choose either mode as required.

- **Authentication mode 1**: AK/SK authentication

  ```
  gaussdb=# \key_info keyType=huawei_kms,kmsProjectId={Project ID},ak={AK},sk={SK}
  ```

  Parameters: For details about how to obtain parameters, including the project ID, AK, and SK, see the master key generation phase.

  Example: \key_info keyType=huawei_kms,kmsProjectId=0b59929e8100268a2f22c01429802728,ak=XMAUMJY******DFWLQW,sk=ga6rO8lx1Q4uB*********2gf80muIzUX

- **Authentication mode 2**: Account/Password authentication

  ```
  gaussdb=# \key_info keyType=huawei_kms,iamUrl={IAM server address},iamUser={IAM username},iamPassword={IAM user password},iamDomain={Account name},kmsProject={Project}
  ```

  Parameters: For details about how to obtain related parameters, including the IAM server address, IAM username, IAM user password, account name, and project, see the master key generation phase.

  Example: \key_info keyType=huawei_kms,iamUrl=https://iam.*example*.com/v3/auth/tokens,iamUser=test,iamPassword=*********,iamDomain=test_account,kmsProject=xxx

Step 3 Define master keys.

In the master key generation phase, the KMS has generated and stored master keys. Running this syntax only stores the master key information into the database for future access. For details about the syntax format, see "SQL Reference > SQL Syntax > C > CREATE CLIENT MASTER KEY" in *Reference*.

```
gaussdb=# CREATE CLIENT MASTER KEY cmk2 WITH ( KEY_STORE = sdf_kms, KEY_PATH = '{KEK ID}');
gaussdb=# CREATE CLIENT MASTER KEY cmk1 WITH ( KEY_STORE = huawei_kms , KEY_PATH = '{KMS server address}|{Key ID}', ALGORITHM = AES_256);
CREATE CLIENT MASTER KEY
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Parameters: For details about how to obtain related parameters, including the KMS server address and key ID, see the master key generation phase.

  Example of *KEY_PATH*: https://kms.cn-north-4.myhuaweicloud.com/v1.0/0b59929e8100268a2f22c01429802728/kms/9a262917-8b31-41af-a1e0-a53235f32de9

Step 4 Define column keys.

Column keys are encrypted by the master keys defined in the previous step. For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE COLUMN ENCRYPTION KEY" in *Reference*.

```
gaussdb=# CREATE COLUMN ENCRYPTION KEY cek1 WITH VALUES (CLIENT_MASTER_KEY = cmk1,
ALGORITHM  = AES_256_GCM);
gaussdb=# CREATE COLUMN ENCRYPTION KEY cek2 WITH VALUES (CLIENT_MASTER_KEY = cmk2,
ALGORITHM = SM4_HMAC_SM3);
```

Step 5 Define an encrypted table.

In this example, the **name** and **credit_card** columns in the table are specified as encrypted columns by using syntax.

```
gaussdb=# CREATE TABLE creditcard_info (
  id_number int,
  name text encrypted with (column_encryption_key = cek1, encryption_type = DETERMINISTIC),
  credit_card varchar(19) encrypted with (column_encryption_key = cek1, encryption_type =
DETERMINISTIC));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id_number' as the distribution column by
default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

Step 6 Perform other operations on the encrypted table.

```
-- Write data to the encrypted table.
gaussdb=# INSERT INTO creditcard_info VALUES (1,'joe','6217986500001288393');
INSERT 0 1
gaussdb=# INSERT INTO creditcard_info VALUES (2, 'joy','6219985678349800033');
INSERT 0 1

-- Query data from the encrypted table.
gaussdb=# select * from creditcard_info where name = 'joe';
 id_number | name |     credit_card
-----------+------+---------------------
         1 | joe  | 6217986500001288393

-- Update data in the encrypted table.
gaussdb=# update creditcard_info set credit_card = '80000000011111111' where name = 'joy';
UPDATE 1

-- Add an encrypted column to the table.
gaussdb=# ALTER TABLE creditcard_info ADD COLUMN age int ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = cek1, ENCRYPTION_TYPE = DETERMINISTIC);
ALTER TABLE

-- Drop an encrypted column from the table.
gaussdb=# ALTER TABLE creditcard_info DROP COLUMN age;
ALTER TABLE

-- Query master key information from the system catalog.
gaussdb=# SELECT * FROM gs_client_global_keys;
 global_key_name | key_namespace | key_owner | key_acl |        create_date
-----------------+---------------+-----------+---------+---------------------------
 cmk1            |          2200 |        10 |         | 2021-04-21 11:04:00.656617
(1 rows)

-- Query column key information from the system catalog.
gaussdb=# SELECT column_key_name,column_key_distributed_id ,global_key_id,key_owner  FROM
gs_column_keys;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 column_key_name | column_key_distributed_id | global_key_id | key_owner
-----------------+---------------------------+---------------+-----------
 cek1            |                760411027 |         16392 |        10
(1 rows)

-- View the meta information of columns in the table.
gaussdb=# \d creditcard_info
        Table "public.creditcard_info"
   Column    |       Type        | Modifiers
-------------+-------------------+------------
 id_number   | integer           |
 name        | text              |  encrypted
 credit_card | character varying |  encrypted
```

Step 7 Clean up.

```
-- Drop the encrypted table.
gaussdb=# DROP TABLE creditcard_info;
DROP TABLE

-- Drop the column keys.
gaussdb=# DROP COLUMN ENCRYPTION KEY cek1;
DROP COLUMN ENCRYPTION KEY
gaussdb=# DROP COLUMN ENCRYPTION KEY cek2;
DROP COLUMN ENCRYPTION KEY

-- Drop the master keys.
gaussdb=# DROP CLIENT MASTER KEY cmk1;
DROP CLIENT MASTER KEY
gaussdb=# DROP CLIENT MASTER KEY cmk2;
DROP CLIENT MASTER KEY
```

----End

7.3.3 Using JDBC to Operate on an Encrypted Database

Configuring the JDBC Driver

1. Obtain the JDBC driver package. For details about how to obtain and use the JDBC driver, see **Obtaining the Driver JAR Package and Configuring the JDK Environment**.

   Encrypted databases support these JDBC driver packages: **gscejdbc.jar**, **gaussdbjdbc.jar**, **opengaussjdbc-ce.jar**, **gsjdbc4.jar**, and **opengaussjdbc.jar**.

   - **gscejdbc.jar** (supported only on EulerOS): The main class name is **com.huawei.gaussdb.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:gaussdb**. This driver package is recommended in encrypted scenarios. The Java code examples in this section use the **gscejdbc.jar** package by default.
   - **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:gaussdb**. Since this driver package does not include the encryption/decryption dependency libraries required by an encrypted database, you need to manually set the environment variable *LD_LIBRARY_PATH*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> An encrypted database also supports other JDBC compatibility driver packages: **opengaussjdbc-ce.jar**, **gsjdbc4.jar** and **opengaussjdbc.jar**.
>
> - **opengaussjdbc-ce.jar**: The main class name is **com.huawei.opengauss.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:opengauss**. Currently, it is supported only on EulerOS. This driver package is recommended for encrypted databases with a URL prefix of **jdbc:opengauss**. It includes the encryption/decryption dependency libraries required by encrypted databases.
> - **gsjdbc4.jar**: The main class name is **org.postgresql.Driver**, and the URL prefix of the database connection is **jdbc:postgresql**.
> - **opengaussjdbc.jar**: The main class name is **com.huawei.opengauss.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:opengauss**.

2. Set *LD_LIBRARY_PATH*.

   Before using JDBC driver packages in encrypted scenarios, you need to set the environment variable *LD_LIBRARY_PATH*. For example:

   ```
   export LD_LIBRARY_PATH=/tmp:$LD_LIBRARY_PATH
   ```

   – When **gscejdbc.jar** or **opengaussjdbc-ce.jar** is used, the contained dependency libraries required by an encrypted database are automatically copied to the specified path. They are loaded when you connect to the database with the encrypted function enabled.

   > **NOTE**
   >
   > When **gscejdbc.jar** or **opengaussjdbc-ce.jar** is used, JVM depends on the libstdc++ library to load class files. If the encrypted function is enabled, the system automatically copies the dynamic libraries (including the libstdc++ library) on which the encrypted database depends to the path specified in *LD_LIBRARY_PATH*. If the version of a dependency library does not match that of the existing system library, only the dependency library is deployed during the first running. After the dependency library is called again, it can be used normally.

   – When using **gaussdbjdbc.jar**, **opengaussjdbc.jar**, or **gsjdbc4.jar**, you need to decompress **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_libpq.tar.gz** to the specified directory, and add the directory of the **lib** folder to the environment variable *LD_LIBRARY_PATH*.

NOTICE

To use the JDBC driver package in the full-encryption scenario, you must have the System.loadLibrary permission as well as the read and write permissions on files in the first-priority path of the environment variable *LD_LIBRARY_PATH*. You are advised to use an independent directory to store the full-encryption dependency library. If **java.library.path** is specified during execution, the value must be the same as the first-priority path of *LD_LIBRARY_PATH*.

Executing SQL Statements

Generate master keys and confirm their access parameters before executing the SQL statements in this section.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

This section uses a complete execution process as an example to describe how to use the encrypted database syntax, including three phases: DDL statement execution, DML statement execution, and cleanup.

For JDBC development operations consistent with non-encrypted scenarios, refer to **Development Based on JDBC**.

- Connection parameters of an encrypted database

  enable_ce: string type. If **enable_ce** is not set, full encryption is disabled. When **enable_ce** is set to **1**, the system enables the basic capability: encrypted equality queries.

```
// The following uses the gscejdbc.jar driver as an example. If other driver packages are used, you only need to change the driver class name and the URL prefix of the database connection.
// gsjdbc4.jar: The main class name is org.postgresql.Driver, and the URL prefix of the database connection is jdbc:postgresql.
// opengaussjdbc-ce.jar: The main class name is com.huawei.opengauss.jdbc.Driver, and the URL prefix of the database connection is jdbc:opengauss.
// opengaussjdbc.jar: The main class name is com.huawei.opengauss.jdbc.Driver, and the URL prefix of the database connection is jdbc:opengauss.
// gscejdbc.jar: The main class name is com.huawei.gaussdb.jdbc.Driver, and the URL prefix of the database connection is jdbc:gaussdb.
// gaussdbjdbc.jar: The main class name is com.huawei.gaussdb.jdbc.Driver, and the URL prefix of the database connection is jdbc:gaussdb.

public static void main(String[] args) {
    // Driver class.
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    // Database connection descriptor. If enable_ce is set to 1, encrypted equality query (a basic capability) is supported.
    String sourceURL = "jdbc:gaussdb://127.0.0.1:8000/postgres?enable_ce=1";
    // Set the username and password in the environment variables USER and PASSWORD, respectively.
    String username = System.getenv("USER");
    String passwd = System.getenv("PASSWORD");
    Connection conn = null;
    try {
        // Load the driver.
        Class.forName(driver);
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
        // Create a statement object.
        Statement stmt = conn.createStatement();

       // Set parameters for accessing master keys.
        // Two modes are provided here. Select either of them.
        // Authentication mode 1: AK/SK authentication (For details about how to obtain related parameters, including the project ID, AK, and SK, see the master key generation phase.)
        conn.setClientInfo("key_info", "keyType=huawei_kms, kmsProjectId={Project ID}, ak={AK}, sk={SK}");

        /* Example:
            conn.setClientInfo("key_info", "keyType=huawei_kms,kmsProjectId=0b59929e8100268a2f22c01429802728," +
                "ak=XMAUMJY******DFWLQW, sk=ga6rO8lx1Q4uB********2gf80muIzUX,");
        */
        // Authentication mode 2: Account/Password authentication (For details about how to obtain related parameters, including the IAM server address, IAM username, IAM user password, account name, and project, see the master key generation phase.)
        conn.setClientInfo("key_info", "keyType=huawei_kms," +
            "iamUrl={IAM server address}," +
            "iamUser={IAM username}," +
            "iamPassword={IAM user password}," +
            "iamDomain={Account name}," +
            "kmsProject={Project}");
        /* Example:
        conn.setClientInfo("key_info", "keyType=huawei_kms," +
                "iamUrl=https://iam.example.com/v3/auth/tokens," +
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                "iamUser=test," +
                "iamPassword=*********," +
                "iamDomain=test_account," +
                "kmsProject=xxx");
        */

        // Define the master key. cmk1 is the master key name, which can be customized.
        // For details about how to obtain the KMS server address and key ID, see the master key generation phase.
        int rc = stmt.executeUpdate("CREATE CLIENT MASTER KEY ImgCMK1 WITH ( KEY_STORE = huawei_kms , KEY_PATH = '{KMS server address}|{Key ID}', ALGORITHM = AES_256);");

        /*
            Example of KEY_PATH: https://kms.cn-north-4.myhuaweicloud.com/v1.0/0b59929e8100268a2f22c01429802728/kms/9a262917-8b31-41af-a1e0-a53235f32de9
            Explanation: In the master key generation phase, the KMS has generated and stored master keys. Running this syntax only stores the master key information into the database for future access.
            Note: For details about the KEY_PATH format, see "SQL Reference > SQL Syntax > C > CREATE CLIENT MASTER KEY" in Reference.
        */

        // Define a column encryption key: The column key is encrypted by the master key created in the previous step. For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE COLUMN ENCRYPTION KEY" in Reference.
        int rc2 = stmt.executeUpdate("CREATE COLUMN ENCRYPTION KEY ImgCEK1 WITH VALUES (CLIENT_MASTER_KEY = ImgCMK1, ALGORITHM  = AES_256_GCM);");
        // Define an encrypted table.
        int rc3 = stmt.executeUpdate("CREATE TABLE creditcard_info (id_number int, name varchar(50) encrypted with (column_encryption_key = ImgCEK1, encryption_type = DETERMINISTIC),credit_card varchar(19) encrypted with (column_encryption_key = ImgCEK1, encryption_type = DETERMINISTIC));");
        // Insert data.
        int rc4 = stmt.executeUpdate("INSERT INTO creditcard_info VALUES (1,'joe','6217986500001288393');");
        // Query the encrypted table.
        ResultSet rs = null;
        rs = stmt.executeQuery("select * from creditcard_info where name = 'joe';");
        // Drop the encrypted table.
        int rc5 = stmt.executeUpdate("DROP TABLE IF EXISTS creditcard_info;");
        // Drop the CEK.
        int rc6 = stmt.executeUpdate("DROP COLUMN ENCRYPTION KEY IF EXISTS ImgCEK1;");
        // Drop the CMK.
        int rc7 = stmt.executeUpdate("DROP CLIENT MASTER KEY IF EXISTS ImgCMK1;");
        // Close the statement object.
        stmt.close();
        // Close the connection.
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
}
```

NOTE

- When JDBC is used to perform operations on an encrypted database, one database connection object corresponds to one thread. Otherwise, conflicts may occur due to thread changes.
- When JDBC is used to perform operations on an encrypted database, different connections change the encrypted configuration data. The client calls the isValid method to ensure that the connections can hold the changed encrypted configuration data. In this case, the **refreshClientEncryption** parameter must be set to **1** (default value). In a scenario where a single client performs operations on encrypted data, the **refreshClientEncryption** parameter can be set to **0**.

- Example of Calling the isValid Method to Refresh the Cache

```
// Create a connection named conn1.
Connection conn1 = DriverManager.getConnection("url","user","password");
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Create a CMK in another connection, conn2.
...
// conn1 calls the isValid method to refresh its own key cache.
try {
 if (!conn1.isValid(60)) {
  System.out.println("isValid Failed for connection 1");
 }
} catch (SQLException e) {
 e.printStackTrace();
       return null;
}
```

Decrypting the Encrypted Equality Ciphertext

Database connection APIs now include a new decryption API (PgConnection type), which can decrypt encrypted equality ciphertext in fully-encrypted databases. After decryption, it returns the plaintext value, identifies the corresponding ciphertext column based on **schema.table.column**, and returns its original data type.

Table 7-1 New com.huawei.gaussdb.jdbc.jdbc.PgConnection API

Method Name	Return Type	Support JDBC4
decryptData(String ciphertext, Integer len, String schema, String table, String column)	ClientLogicDecryptResult	Yes

Parameters:

- **ciphertext**

  Ciphertext to be decrypted.

- **len**

  Ciphertext length. If the value is less than the actual ciphertext length, decryption fails.

- **schema**

  Name of the schema to which the encrypted column belongs.

- **table**

  Name of the table to which the encrypted column belongs.

- **column**

  Name of the column to which the encrypted column belongs.

  NOTE

  Decryption is successful in the following scenarios, but is not recommended:

  - The ciphertext length input parameter is longer than the actual ciphertext.
  - **schema.table.column** points to another encrypted column. In this case, the original data type of the encrypted column is returned.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-2 New com.huawei.gaussdb.jdbc.jdbc.clientlogic.ClientLogicDecryptResult API

Method Name	Return Type	Description	Support JDBC4
isFailed()	Boolean	Specifies whether the decryption fails. If the decryption fails, **True** is returned. Otherwise, **False** is returned.	Yes
getErrMsg()	String	Obtains error information.	Yes
getPlaintext()	String	Obtains the decrypted plaintext.	Yes
getPlaintextSize()	Integer	Obtains the length of the decrypted plaintext.	Yes
getOriginalType()	String	Obtains the original data type of the encrypted column.	Yes

```
// After the ciphertext is obtained through non-encrypted connection or logical decoding, this API can be used to decrypt the ciphertext.
import com.huawei.gaussdb.jdbc.jdbc.PgConnection;
import com.huawei.gaussdb.jdbc.jdbc.clientlogic.ClientLogicDecryptResult;

// conn is an encrypted connection.
// Call the decryptData method of PgConnection to decrypt the ciphertext, locate the encrypted column to which the ciphertext belongs based on the column name, and return the original data type.
ClientLogicDecryptResult decrypt_res = null;
decrypt_res = ((PgConnection)conn).decryptData(ciphertext, ciphertext.length(), schemaname_str,
        tablename_str, colname_str);
// Check whether the decryption of the returned result class is successful. If the decryption fails, obtain the error information. If the decryption is successful, obtain the plaintext, length, and original data type.
if (decrypt_res.isFailed()) {
    System.out.println(String.format("%s\n", decrypt_res.getErrMsg()));
} else {
    System.out.println(String.format("decrypted plaintext: %s size: %d type: %s\n", decrypt_res.getPlaintext(),
        decrypt_res.getPlaintextSize(), decrypt_res.getOriginalType()));
}
```

Executing Prepared SQL Statements on an Encrypted Table

```
// Create a prepared statement object by calling the prepareStatement method of Connection.
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO creditcard_info VALUES (?, ?, ?);");
// Set parameters by calling the setShort method of PreparedStatement.
pstmt.setInt(1, 2);
pstmt.setString(2, "joy");
pstmt.setString(3, "6219985678349800033");
// Execute prepared statements by calling the executeUpdate method of PreparedStatement.
int rowcount = pstmt.executeUpdate();
// Close the prepared statement object by calling the close method of PreparedStatement.
pstmt.close();
```

Batch Processing on an Encrypted Table

```
// Create a prepared statement object by calling the prepareStatement method of Connection.
Connection conn = DriverManager.getConnection("url","user","password");
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO creditcard_info (id_number, name,
credit_card) VALUES (?,?,?)");
// Call the setShort method to set parameters for each data item, and call addBatch to confirm that the
setting is complete.
int loopCount = 20;
 for (int i = 1; i < loopCount + 1; ++i) {
      pstmt.setInt(1, i);
      pstmt.setString(2, "Name " + i);
      pstmt.setString(3, "CreditCard " + i);
      // Add row to the batch.
      pstmt.addBatch();
}
// Perform batch processing by calling the executeBatch method of PreparedStatement.
int[] rowcount = pstmt.executeBatch();
// Close the prepared statement object by calling the close method of PreparedStatement.
pstmt.close();
```

7.3.4 Backward Compatibility and Security Enhancement

Backward Compatibility

In the preceding sections, you can use **key_info** to set parameters for accessing an external key manager.

1. When gsql is used, you can use the meta-command **\key_info** *xxx*.
2. When JDBC is used, you can use the connection parameter **conn.setProperty("key_info", "*xxx*")**.

To ensure backward compatibility, you can set the master key access parameters through environment variables.

CAUTION

If you use an encrypted database for the first time, skip the following steps. If you have used any of the following methods to configure an encrypted database, you are advised to configure **key_info** instead.

To use system-level environment variables, perform the following steps:

```
export HUAWEI_KMS_INFO='iamUrl=https://iam.{Project}.myhuaweicloud.com/v3/auth/
tokens,iamUser={IAM username},iamPassword={IAM user password},iamDomain={Account
name},kmsProject={Project}'
# In this method, OS logs may record sensitive information from environment variables. It is crucial to clean
up promptly.
```

You can also set process-level environment variables using the standard library API. The methods for setting process-level environment variables in different languages are as follows:

1. C/C++

   ```
   setenv("HIS_KMS_INFO", "xxx");
   ```

2. Go

   ```
   os.Setenv("HIS_KMS_INFO", "xxx");
   ```

Verifying the External KMS Identity

When the database driver accesses Huawei Cloud KMS, use a CA certificate to verify the validity of the key server during the establishment of HTTPS connections between the database driver and the KMS. This verification prevents attackers

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

from masquerading as the KMS. Therefore, you need to configure a CA certificate in advance. Without this certificate, the KMS identity cannot be verified. This section describes how to download and configure a CA certificate.

Configuration Method

Add certificate-related **key_info** parameters.

- When gsql is used:

```
gaussdb=# \key_info keyType=huawei_kms,iamUrl=https://iam.example.com/v3/auth/tokens,iamUser={IAM username},iamPassword={IAM user password},iamDomain={Account name},kmsProject={Project},iamCaCert=/Path/IAM CA certificate file,kmsCaCert=/Path/KMS CA certificate file


gaussdb=# \key_info keyType=huawei_kms,kmsProjectId={Project ID},ak={AK},sk={SK},kmsCaCert=/Path/KMS CA certificate file
```

- When JDBC is used:

```
conn.setProperty("key_info", "keyType=huawei_kms," +
    "iamUrl=https://iam.{example.com/v3/auth/tokens," +
    "iamUser={IAM username}," +
    "iamPassword={IAM user password}," +
    "iamDomain={Account name}," +
    "kmsProject={Project}," +
    "iamCaCert=/Path/IAM CA certificate file," +
    "kmsCaCert=/Path/KMS CA certificate file");

conn.setProperty("key_info", "keyType=huawei_kms, kmsProjectId={Project ID}, ak={AK}, sk={SK}, kmsCaCert=/Path/KMS CA certificate file");
```

Obtaining a Certificate

Most browsers automatically download a CA certificate of a website and provide the certificate export function. Although many websites provide the function of automatically downloading CA certificates, these certificates may be unavailable due to proxy or gateway settings in the local environment. Therefore, you are advised to use a browser to download the CA certificate, as follows:

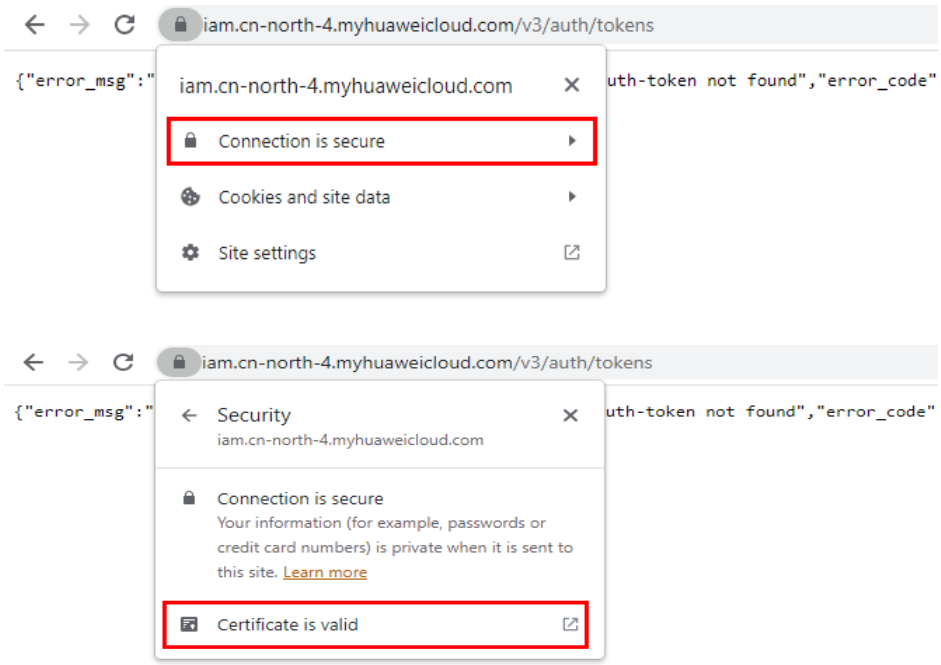
CAUTION

The RESTful API is used to access the KMS. When you enter the URL of the API in the address bar of the browser, ignore the failure page in **Step 2**. The browser has automatically downloaded the CA certificate in advance even if the failure page is displayed.

Step 1 Open a browser. In the Huawei Cloud scenario, enter the addresses of the IAM and KMS servers. For details about how to obtain these addresses, see **Master Key Generation Phase**.

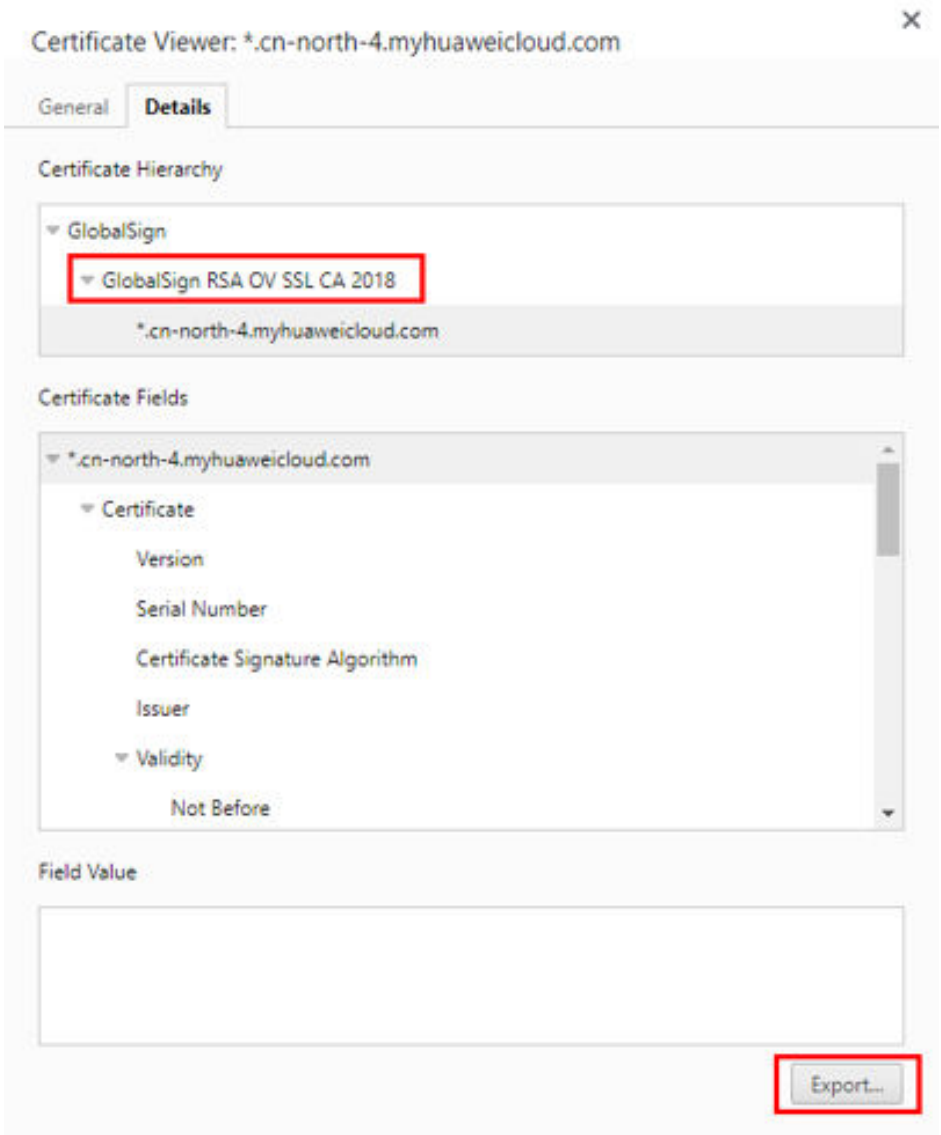
Step 2 Search for a certificate: Each time you enter a domain name, find the SSL connection information and click the information to view the certificate content.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.



Step 3 Export the certificate: On the **Certificate Viewer** page, certificates may be classified into multiple levels. You only need to select the upper-level certificate of the domain name and click **Export** to generate a certificate file, that is, the required certificate file.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.



Step 4 Upload the certificate: Upload the exported certificate to the application and set the preceding parameters.

----End

7.3.5 Encrypted Functions and Stored Procedures

In the current version, only encrypted functions and stored procedures in SQL or PL/pgSQL are supported. Because users are unaware of the creation and execution of functions or stored procedures in an encrypted stored procedure, the syntax has no difference from that of non-encrypted functions and stored procedures.

For details about function/stored procedure syntax, see **User-Defined Functions** and "Stored Procedure" in *Reference*.

The gs_encrypted_proc system catalog is added to store the original return types of parameters in encrypted equality query functions and stored procedures.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about the fields in the system catalog, see "System Catalogs and System Views > System Catalogs > Encrypted Equality Query > GS_ENCRYPTED_PROC" in *Reference*.

Creating and Executing a Function or Stored Procedure That Involves Encrypted Columns

Step 1 Create a key. For details, see **Using gsql to Operate on an Encrypted Database**.

Step 2 Create an encrypted table.

```
gaussdb=# CREATE TABLE creditcard_info (
  id_number int,
  name  text,
  credit_card varchar(19) encrypted with (column_encryption_key = cek1, encryption_type =
DETERMINISTIC)
 ) with (orientation=row) distribute by hash(id_number);
CREATE TABLE
```

Step 3 Insert data.

```
gaussdb=# insert into creditcard_info values(1, 'Avi', '1234567890123456');
INSERT 0 1
gaussdb=# insert into creditcard_info values(2, 'Eli', '2345678901234567');
INSERT 0 1
```

Step 4 Create a function supporting encrypted equality query.

```
gaussdb=# CREATE FUNCTION f_encrypt_in_sql(val1 text, val2 varchar(19)) RETURNS text AS 'SELECT
name from creditcard_info where name=$1 or credit_card=$2 LIMIT 1' LANGUAGE SQL;
CREATE FUNCTION
gaussdb=# CREATE FUNCTION f_encrypt_in_plpgsql (val1 text, val2 varchar(19), OUT c text) AS $$
 BEGIN
 SELECT into c name from creditcard_info where name=$1 or credit_card =$2 LIMIT 1;
 END; $$
 LANGUAGE plpgsql;
CREATE FUNCTION
```

Step 5 Execute the function.

```
gaussdb=# SELECT f_encrypt_in_sql('Avi','1234567890123456');
 f_encrypt_in_sql
------------------
 Avi
(1 row)

gaussdb=# SELECT f_encrypt_in_plpgsql('Avi', val2=>'1234567890123456');
 f_encrypt_in_plpgsql
----------------------
 Avi
(1 row)
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

1. Because the query, that is, the dynamic query statement executed in a function or stored procedure, is compiled during execution, the table and column names in the function or stored procedure cannot remain unknown during the creation phase. Neither individual table/column names nor their combinations can be passed as input parameters.
2. For the dynamic query statement executed in a function or stored procedure, EXECUTE 'query' must not contain the data values to be encrypted.
3. The function does not support scenarios where RETURN QUERY EXECUTE 'query' returns ciphertext data while the function's declared return type is plaintext.
4. Among the **RETURNS**, **IN**, and **OUT** parameters, encrypted and non-encrypted parameters cannot be used together. Although the parameter types are all original, the actual types are different.
5. Among advanced package APIs, those printing data on the server, such as dbe_output.print_line(), skip the decrypt process by design. This is because when an encrypted data type is forcibly converted into its original plaintext equivalent, these APIs print the default value of that data type.
6. In the current version, functions and stored procedures accept only SQL and PL/pgSQL as their **LANGUAGE**. They do not support other procedural languages such as C or Java.
7. Other functions or stored procedures for querying encrypted columns cannot be executed in a function or stored procedure.
8. The current version does not support assignment of default values to variables using the DEFAULT keyword or within the DECLARE block. Additionally, decryption of return variables in the DECLARE block, as well as decryption of declarations between IS/AS and BEGIN, are not supported. Users can use input and output parameters instead when executing functions.
9. gs_dump cannot back up and then restore functions involving encrypted columns.
10. Keys cannot be created in functions or stored procedures.
11. Encrypted functions and stored procedures do not support triggers.
12. Encrypted equality query functions and stored procedures do not support escaping PL/pgSQL syntax. To address this limitation, you can use CREATE FUNCTION AS $$*Syntax body*$$ instead of CREATE FUNCTION AS '*Syntax body*'.
13. After encrypted column definitions are modified within an encrypted equality query function or stored procedure, those columns cannot be used immediately for encryption operations. For example, composite operations—such as creating an encrypted table, adding encrypted columns, and inserting data into the table within a stored procedure—are not supported. This limitation exists because functions and stored procedures execute on the server, and the client cannot determine whether a cache refresh is required. To enable encryption operations on those columns, you must either close the connection or trigger a refresh of the client's encrypted column cache.
14. Functions and stored procedures cannot be created using encrypted data types (byteawithoutorderwithequalcol, byteawithoutordercol, _byteawithoutorderwithequalcol, or _byteawithoutordercol).
15. If an encrypted function returns a value of an encrypted type, the result cannot be an uncertain row type, for example, **RETURN [SETOF] RECORD**. You can replace it with a definite row type, for example, **RETURN TABLE(columnname typename[, ...])**.
16. When an encrypted function is created, the OID of the encrypted column corresponding to a parameter is added to the gs_encrypted_proc system catalog. Therefore, if a table with the same name is deleted and created again, the encrypted function may become invalid and you need to create the encrypted function again.

7.4 Transparent Data Encryption

Transparent data encryption (TDE) provides the function of encrypting and storing table-level data. After you use the syntax provided by this feature to create an

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

encrypted table, the database automatically encrypts the table data before writing it to disk. The database also automatically decrypts the data after reading it from disk. The syntax for inserting, updating, querying, and deleting data in encrypted tables is identical to that used for non-encrypted tables.

NOTE

- Before using TDE, enable it first by referring to "Account and Network Security > Enabling Transparent Data Encryption for a GaussDB Instance" in *User Guide*.

Checking Basic TDE Configurations

Step 1 Check whether TDE is enabled.

If **enable_tde** is set to **on**, TDE is enabled. If **enable_tde** is set to **off**, TDE is disabled.

```
gaussdb=# SHOW enable_tde;
 enable_tde
------------
 on
(1 row)
```

Step 2 Check whether the KMS access parameters have been set.

If the **tde_key_info** parameter is empty, no setting is performed. If the **tde_key_info** parameter is not empty, the setting is done.

```
gaussdb=# show tde_key_info;
      tde_key_info
-------------------------
 keyType=...
```

----End

Operating on an Encrypted Table

Step 1 Create an encrypted table.

When creating a table, you can set **enable_tde** to **on** in the WITH clause to encrypt the table.

By default, the database uses the AES_128_CTR algorithm to encrypt tables. To use other algorithms, set the **encrypt_algo** parameter.

```
gaussdb=# CREATE TABLE t1 (c1 INT, c2 TEXT) WITH (enable_tde = on);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (c1 INT, c2 TEXT) WITH (enable_tde = on, encrypt_algo = 'SM4_CTR');
CREATE TABLE
```

Step 2 View basic information about the encrypted table.

Basic information about encrypted tables is stored in the **reloptions** column in the pg_class system catalog. **dek_cipher** indicates the data key ciphertext, which is automatically generated by the database and encrypted by KMS. Each encrypted table has an independent data key.

```
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 't1';
 relname | reloptions
---------
+----------------------------------------------------------------------------------------------------------------------------
 t1      |
{orientation=row,enable_tde=on,encrypt_algo=AES_128_CTR,compression=no,storage_type=USTORE,key_type
=...,dek_cipher=...
```

Execute the following statement to query all encrypted tables in the database:

```
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE reloptions::text LIKE '%enable_tde=on%';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------
 t1      |
{orientation=row,enable_tde=on,encrypt_algo=AES_128_CTR,compression=no,storage_type=USTORE,key_type
=...,dek_cipher=...
```

Step 3 Write data to the encrypted table.

The syntax for operating on encrypted tables is the same as that for non-encrypted tables. The database automatically encrypts data in the encrypted table only before writing it to disk.

```
gaussdb=# INSERT INTO t1 VALUES (1, 'tde plain 123');
INSERT 0 1
```

Step 4 Query data from the encrypted table.

For authorized users, the syntax for querying encrypted tables is the same as that for querying non-encrypted tables. Encryption and decryption operations are automatically implemented by the database. If an attacker bypasses the database and reads the encrypted table's data file directly from disk, they will find that all data in the file is encrypted.

```
gaussdb=# SELECT * FROM t1;
 c1 |      c2
----+---------------
  1 | tde plain 123
(1 row)
```

Step 5 Rotate the key of the encrypted table.

To improve security, you are advised to periodically use the following syntax to rotate the data key of the encrypted table, that is, use the new key to encrypt data:

```
gaussdb=# ALTER TABLE t1 ENCRYPTION KEY ROTATION;
ALTER TABLE
```

After the key is rotated, the database can still decrypt the data encrypted by the old key.

Step 6 Convert between encrypted and non-encrypted tables.

TDE supports the conversion between encrypted tables and non-encrypted tables. After each conversion, you are advised to manually run the **VACUUM FULL** *tablename* command to synchronously convert all data in the table.

```
gaussdb=# CREATE TABLE t3 (c1 INT, c2 TEXT);
CREATE TABLE
gaussdb=# ALTER TABLE t3 SET (enable_tde = on);
ALTER TABLE
gaussdb=# VACUUM FULL t3;
VACUUM
gaussdb=# ALTER TABLE t3 SET (enable_tde = off);
ALTER TABLE
gaussdb=# VACUUM FULL t3;
VACUUM
```

Step 7 Drop the encrypted table.

```
gaussdb=# DROP TABLE IF EXISTS t1, t2, t3;
DROP TABLE
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operating on an Encrypted Index

Step 1 Create an encrypted table.

Create an encrypted base table.

```
gaussdb=# CREATE TABLE t1 (c1 INT, c2 TEXT) WITH (enable_tde = on);
CREATE TABLE
```

Step 2 Create an encrypted index.

The operation is the same as that of creating an encrypted table. You can set **enable_tde** to **on** in the WITH clause to create an encrypted index.

The index uses the same encryption algorithm and key as the base table. When key rotation is performed on the base table, the new key also applies to the index.

```
gaussdb=# CREATE INDEX i1 ON t1(c2) WITH (enable_tde = on);
CREATE INDEX
```

Step 3 View basic information about the encrypted indexes.

Similar to encrypted tables, indexes also have their basic information stored in the **reloptions** column in the pg_class system catalog. Their parameters, such as **dek_cipher** and **encrypt_algo**, are the same as those of base tables.

```
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 'i1';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------
 i1      |
{orientation=row,enable_tde=on,encrypt_algo=AES_128_CTR,compression=no,storage_type=USTORE,key_type
=...,dek_cipher=...
```

Step 4 Convert between an encrypted index and a non-encrypted index.

TDE allows you to convert an encrypted index to a non-encrypted one, and vice versa.

```
gaussdb=# CREATE TABLE t2 (c1 INT, c2 TEXT) WITH (enable_tde = on);
ALTER TABLE
gaussdb=# CREATE INDEX i2 ON t2(c2);
CREATE INDEX
gaussdb=# ALTER INDEX i2 SET (enable_tde = on);
ALTER INDEX
gaussdb=# ALTER INDEX i2 SET (enable_tde = off);
ALTER INDEX
```

Step 5 Automatically encrypt an index.

By default, an encrypted index can be created only when the **enable_tde** parameter is set. When the GUC parameter **tde_index_default_encrypt** is set to **on** and an index is created based on an encrypted table, the database automatically converts the index to an encrypted index. For example:

```
gaussdb=# CREATE TABLE t3 (c1 INT, c2 TEXT) WITH (enable_tde = on);
ALTER TABLE
gaussdb=# CREATE INDEX i3 ON t3(c2);
CREATE INDEX
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 'i3';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------
 i3      |
{orientation=row,enable_tde=on,encrypt_algo=AES_128_CTR,compression=no,storage_type=USTORE,key_type
=...,dek_cipher=...
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Explanation: Although i3 is not explicitly specified as an encrypted index, the database automatically converts i3 to an encrypted index because tde_index_default_encrypt is set to on and the base table t3 is an encrypted table.
```

Step 6 Drop the encrypted table and index.

```
gaussdb=# DROP TABLE IF EXISTS t1, t2, t3;
DROP TABLE
```

----End

Encrypting Xlogs and Undo Logs

Step 1 Create a TDE table.

```
gaussdb=# CREATE TABLE t1 (c1 INT, c2 TEXT) WITH (enable_tde = on);
CREATE TABLE
```

Step 2 Record the Xlog position.

```
gaussdb=# SELECT pg_xlogfile_name_offset((SELECT pg_current_xlog_location()));
     pg_xlogfile_name_offset
-----------------------------------
 (00000001000000000000000D,3121400)
(1 row)
```

Step 3 Perform operations on the TDE table to generate Xlogs that contain data.

```
gaussdb=# INSERT INTO t1 VALUES (1, 'sssssssssssssss');
INSERT 0 1
```

Step 4 Based on the Xlog position obtained in **Step 2**, run the **hexdump** command to verify that data in the Xlogs has been encrypted. Replace *$DATADIR* with the actual data directory.

```
gaussdb=#  \! hexdump -C $DATADIR/pg_xlog/00000001000000000000000D -s 3121400 -n 2048
002fa0f8  5e 00 00 80 01 00 00 00  00 00 00 00 00 00 00 00  |^...............|
002fa108  d8 9c 2f 0d 00 00 00 00  e0 16 00 00 00 00 00 00  |../.............|
002fa118  26 74 d4 9e ff 38 b8 17  00 00 00 00 00 00 d8 0e  |&t...8..........|
002fa128  00 00 00 00 00 00 52 31  01 00 00 00 00 00 6b 26  |......R1......k&|
002fa138  06 00 00 00 00 00 7a 33  01 00 00 00 00 00 7b 33  |......z3......{3|
002fa148  01 00 00 00 00 00 f0 3d  2f 0d 00 00 00 00 00 00  |.......=/.......|
002fa158  3e 00 00 80 01 00 00 00  ab 33 01 00 00 00 00 00  |>........3......|
002fa168  f8 a0 2f 0d 00 00 00 00  80 16 00 00 00 00 00 00  |../.............|
002fa178  14 79 79 89 ff 18 00 00  00 00 00 10 00 00 00 00  |.yy.............|
002fa188  10 00 00 10 00 00 00 00  00 00 00 00 00 00 00 00  |................|
002fa198  3e 00 00 80 01 00 00 00  ab 33 01 00 00 00 00 00  |>........3......|
002fa1a8  58 a1 2f 0d 00 00 00 00  b0 16 00 00 00 00 00 00  |X./.............|
002fa1b8  6d 09 43 cc ff 18 00 00  00 00 00 10 00 00 00 80  |m.C.............|
002fa1c8  00 00 00 10 00 00 00 00  00 00 00 00 00 00 00 00  |................|
002fa1d8  13 02 00 80 01 00 00 00  ab 33 01 00 00 00 00 00  |.........3......|
002fa1e8  98 a1 2f 0d 00 00 00 00  80 14 00 00 00 00 00 80  |../.............|
002fa1f8  12 61 3e 44 40 60 1d 00  7f 06 00 00 13 32 00 00  |.a>D@`.......2..|
002fa208  57 41 00 00 25 94 24 f9  f6 8d 46 50 8a f7 3f 92  |WA..%.$...FP..?.|
002fa218  98 08 2c 57 a2 fc 16 3d  18 dc 6f 67 9e 8a c8 ba  |..,W...=..og....|
002fa228  a3 7b 58 3f ba 50 d9 6c  52 d2 91 01 75 a7 a4 d9  |.{X?.P.lR...u...|
002fa238  05 86 2d b1 10 aa 1c 46  14 7f 00 00 d3 3a 4a 22  |..-....F.....:J"|
002fa248  e0 55 00 00 7f 06 00 00  09 00 00 01 00 35 a8 44  |.U...........5.D|
002fa258  00 7f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Step 5 Delete data from the **t1** table to generate undo logs. Verify that the undo file has been encrypted. Replace *$DATADIR* with the actual data directory.

Expected result: No plaintext data can be found.

```
gaussdb=# DELETE FROM t1;
DELETE 1
gaussdb=# \q
grep -c 'sssssssssssssss' $DATADIR/undo/permanent/*
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 6 Drop the encrypted table.

```
gaussdb=# DROP TABLE IF EXISTS t1;
DROP TABLE
```

----End

Enabling Asynchronous TDE

Step 1 Check whether asynchronous TDE is enabled.

If **enable_tde_async_encryption** is set to **on**, the function is enabled. If it is set to **off**, the function is disabled.

```
gaussdb=# SHOW enable_tde_async_encryption;
 enable_tde_async_encryption
-----------------------------
 on
(1 row)
```

Step 2 Create multiple TDE tables.

```
gaussdb=# CREATE TABLE t1 (C1 INT, C2 TEXT) WITH (ENABLE_TDE = ON);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (C1 INT, C2 TEXT) WITH (ENABLE_TDE = ON, ENCRYPT_ALGO = 'SM4_CTR');
CREATE TABLE
```

Step 3 View basic information about the encrypted tables.

All the encrypted tables created in the previous step use the same data key (the asynchronous encryption key), labeled as **<async_dek1>**.

```
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 't1';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------
 t1      | {enable_tde=on,encrypt_algo=AES_128_CTR,...,dek_cipher=<async_dek1>
(1 row)
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 't2';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------
 t2      | {enable_tde=on,encrypt_algo=SM4_CTR,...,dek_cipher=<async_dek1>
(1 row)
```

Step 4 Rotate the table-level encryption key.

Note that the table-level data key cannot be rotated when **enable_tde_async_encryption** is enabled.

```
gaussdb=# ALTER TABLE t1 ENCRYPTION KEY ROTATION;
NOTICE:  Unable to rotate key for encrypted table when parameter enable_tde_async_encryption is on.
ALTER TABLE
```

Step 5 Rotate the asynchronous encryption key.

```
gaussdb=# ALTER ASYNC ENCRYPTION KEY ROTATION;
ALTER TDE ROTATE ENCRYPTION KEY
```

Step 6 Create multiple encrypted tables and view basic information about them.

The newly created encrypted tables use the same data key (the new asynchronous encryption key after rotation), labeled as **<async_dek2>**. This key differs from the one created in step 2.

```
gaussdb=# CREATE TABLE t3 (C1 INT, C2 TEXT) WITH (ENABLE_TDE = ON);
CREATE TABLE
gaussdb=# CREATE TABLE t4 (C1 INT, C2 TEXT) WITH (ENABLE_TDE = ON, ENCRYPT_ALGO = 'SM4_CTR');
CREATE TABLE

gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 't3';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------------
 t3      | {enable_tde=on,encrypt_algo=AES_128_CTR,...,dek_cipher=<async_dk2>
(1 row)
gaussdb=# SELECT relname,reloptions FROM pg_class WHERE relname = 't4';
 relname | reloptions
---------
+-----------------------------------------------------------------------------------------------------------------------------
 t4      | {enable_tde=on,encrypt_algo=SM4_CTR,...,dek_cipher=<async_dk2>
(1 row)
```

Step 7 Drop all encrypted tables.

```
gaussdb=# DROP TABLE t1,t2,t3;
DROP TABLE
```

----End

Loading the Kunpeng Accelerator Engine for TDE

Step 1 Verify that KAE 2.0 has been correctly installed on each node in the database cluster and that the database OS user has permission on HiSec 2.0 devices and KAE logs.

NOTE

For details about how to install KAE and configure permissions, visit the **Kunpeng community**.

Step 2 Before installing the cluster, configure environment variables on each node to point to the installation path of the OpenSSL KAE dynamic library. Alternatively, configure environment variables after the cluster is installed. However, you must restart the om_monitor process for the configuration to take effect.

```
export OPENSSL_ENGINES=/usr/local/lib/engines-3.0
```

Step 3 Start the database.

Check the gs_log file. If the following log is recorded, KAE has been successfully loaded to GaussDB:

```
Tde has successfully initialized the KAE engine and fetched the SM4 cipher.
```

----End

Disabling TDE

Set **enable_tde** to **off**. For details, see "Database GUC Parameters > GUC Parameters > Security Configuration" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

Once TDE is enabled, disabling it is not recommended as it can lead to the following issues:

- Encrypted tables cannot be created, and data in existing encrypted tables cannot be decrypted or read.
- If log encryption has been enabled, disabling TDE will cause failure to replay encrypted redo/undo logs. As a result, the cluster may fail to start or node status may become abnormal.
- If encrypted tables or encrypted log data has been backed up, restoration from these backups will fail.

Typical Issues

- Issue 1: The client output or log indicates a curl failure, returning the error message "Couldn't resolve host."

  Cause analysis: The OS cannot map the domain name configured in **tde_key_info** to an IP address.

  Solution: Obtain the IP address of the configured IAM or KMS service. Then, as the **root** user, add domain name resolution rules to the **/etc/hosts** file (**/var/chroot/etc/hosts** in sandbox environments).

  ```
  echo 'xxx.xxx.xxx.xxx hostname' >> /etc/hosts
  ```

- Issue 2: The client output or log displays the error message "failed to find file '$GAUSSHOME/gs_security/tde.key.cipher'."

  Cause analysis: In KMS scenarios, authentication-sensitive information is not encrypted on the primary or standby node.

  Solution: Ensure that authentication-sensitive information has been encrypted on all database nodes. Then restart the database.

- Issue 3: In scenarios using the third-party standard encryption library GM0018, repeated attempts to execute TDE-related DDL/DML statements fail with a message indicating an error in calling HSM APIs.

  Cause analysis: The .so library configurations of the HSM client are incorrect, the HSM server is abnormal, or the semaphore in the node environment is abnormal.

  Solution: Contact O&M personnel to correct the .so library configurations of the HSM client and troubleshoot server faults. Run the **ipcs** command in the node environment to check whether the database user's HSM semaphore has been improperly cleared. After resolving these issues, attempt a restart to restore normal operations.

7.5 Ledger Database

7.5.1 Overview

Context

The ledger database, inspired by blockchain concepts, records each user operation in two history tables: a user history table and the global blockchain table. When

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

you create a tamper-proof user table, the system automatically adds a hash column to store the hash digest of each row. At the same time, it creates a user history table in the blockchain schema to track changes to each data record in the user table. The system also logs all changes to the tamper-proof user table in the global blockchain table. Because the history tables allow only appends and no changes, their records chain together to form a complete change history of the tamper-proof user table.

The name and structure of the user history table are as follows:

Table 7-3 Columns in the user history table blockchain.*<schemaname>*_*<tablename>*_hist

Column Name	Type	Description
rec_num	bigint	Sequence number of a row-level change in the history table.
hash_ins	hash16	Hash value of the data row inserted by an INSERT or UPDATE operation.
hash_del	hash16	Hash value of the data row deleted by a DELETE or UPDATE operation.
pre_hash	hash32	Overall digest of the current user history table.

Table 7-4 Mapping between hash_ins and hash_del

-	hash_ins	hash_del
INSERT	(√) Hash value of the inserted row.	Empty.
DELETE	Empty.	(√) Hash value of the deleted row.
UPDATE	(√) Hash value of the newly inserted data.	(√) Hash value of the row before deletion.

Procedure

Step 1 Create a tamper-proof schema.

For example, create a tamper-proof **ledgernsp** schema.

```
gaussdb=# CREATE SCHEMA ledgernsp WITH BLOCKCHAIN;
```

NOTE

To create a tamper-proof schema or shift from an ordinary schema to a tamper-proof schema, set **enable_ledger** to **on**. The default value of **enable_ledger** is **off**.

Step 2 Create a tamper-proof user table in the tamper-proof schema.

For example, create a tamper-proof user table named **ledgernsp.usertable**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE TABLE ledgernsp.usertable(id int, name text);
```

Check the structures of the tamper-proof user table and the corresponding user history table.

```
gaussdb=# \d+ ledgernsp.usertable;
gaussdb=# \d+ blockchain.ledgernsp_usertable_hist;
```

The result is as follows:

```
gaussdb=# \d+ ledgernsp.usertable;
                    Table "ledgernsp.usertable"
 Column     |  Type   | Modifiers | Storage  | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
 id         | integer |           | plain    |              |
 name       | text    |           | extended |              |
 hash_69dd43   | hash16  |           | plain    |              |
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
History table name: ledgernsp_usertable_hist

gaussdb=# \d+ blockchain.ledgernsp_usertable_hist;
             Table "blockchain.ledgernsp_usertable_hist"
  Column  |  Type  | Modifiers | Storage | Stats target | Description
----------+--------+-----------+---------+--------------+-------------
 rec_num  | bigint |           | plain   |              |
 hash_ins | hash16 |           | plain   |              |
 hash_del | hash16 |           | plain   |              |
 pre_hash | hash32 |           | plain   |              |
Indexes:
    "gs_hist_69dd43_index" PRIMARY KEY, btree (rec_num int4_ops) TABLESPACE pg_default
Has OIDs: no
Distribute By: HASH(rec_num)
Location Nodes: ALL DATANODES
Options: internal_mask=263
```

NOTE

1. In tamper-proof schemas, only row-store tables can be configured as tamper-proof. Temporary tables, foreign tables, unlogged tables, and non-row-store tables lack the tamper-proof attribute.
2. When a tamper-proof table is created, a system column is automatically added for verification purposes. Consequently, a tamper-proof table can have a maximum of 1,599 columns.

Step 3 Modify data in the tamper-proof user table.

For example, perform INSERT, UPDATE, and DELETE on the tamper-proof user table.

```
gaussdb=# INSERT INTO ledgernsp.usertable VALUES(1, 'alex'), (2, 'bob'), (3, 'peter');
INSERT 0 3
gaussdb=# SELECT *, hash_69dd43 FROM ledgernsp.usertable ORDER BY id;
 id | name  |       hash_69dd43
----+-------+------------------
  1 | alex  | 1f2e543c580cb8c5
  2 | bob   | 8fcd74a8a6a4b484
  3 | peter | f51b4b1b12d0354b
(3 rows)

gaussdb=# UPDATE ledgernsp.usertable SET name = 'bob2' WHERE id = 2;
UPDATE 1
gaussdb=# SELECT *, hash_69dd43 FROM ledgernsp.usertable ORDER BY id;
 id | name  |       hash_69dd43
----+-------+------------------
  1 | alex  | 1f2e543c580cb8c5
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 2 | bob2  | 437761affbb7c605
 3 | peter | f51b4b1b12d0354b
(3 rows)

gaussdb=# DELETE FROM ledgernsp.usertable WHERE id = 3;
DELETE 1
gaussdb=# SELECT *, hash_69dd43 FROM ledgernsp.usertable ORDER BY id;
 id | name |       hash_69dd43
----+------+------------------
 1 | alex | 1f2e543c580cb8c5
 2 | bob2 | 437761affbb7c605
(2 rows)
```

Step 4 Drop the table and schema.

If you plan to run examples from other ledger database sections, complete them before performing this step. Otherwise, proceed directly with this step.

```
gaussdb=# DROP TABLE ledgernsp.usertable;
DROP TABLE
gaussdb=# DROP SCHEMA ledgernsp;
DROP SCHEMA
```

----End

7.5.2 Viewing Historical Operation Records in the Ledger

Prerequisites

- You are an audit administrator or a role that has the AUDITADMIN permission.
- The database is running properly, and a series of insert, delete, and update operations have been performed on the tamper-proof database to ensure that ledger operation records are generated in the query period.
- The global blockchain table of each CN operates separately and records only the SQL operations performed on the current CN.

Context

- Only users with the AUDITADMIN attribute can view historical operation records in the ledger. For details about database users and user creation methods, see **Users**.
- To query the global blockchain table **gs_global_chain**, run the following command:

  ```
  SELECT * FROM gs_global_chain;
  ```

  This table contains 10 columns. For details about each column, see "System Catalogs and System Views > System Catalogs > Ledger Database > GS_GLOBAL_CHAIN" in *Reference*.
- When you query the user history table, the command is directly run in the BLOCKCHAIN schema, as follows:

  For example, if the user table **usertable** resides in the ledgernsp schema and the corresponding user history table is **blockchain.ledgernsp_usertable_hist**, you can run the following command:

  ```
  SELECT * FROM blockchain.ledgernsp_usertable_hist;
  ```

  A user history table has four columns. For details about the meaning of each column, see **Table 7-3**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> Typically, the system names a user history table using the **blockchain.**<*schemaname*>_<*tablename*>_**hist** format. If the schema name or table name of the tamper-proof user table is too long, this format may produce a table name that exceeds the length limit. In that case, the system uses the **blockchain.**<*schema_oid*>_<*table_oid*>_**hist** format instead.

Procedure

Step 1 Query records in the global blockchain table.

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

The result is as follows:

```
 blocknum |  dbname  | username |           starttime           | relid |  relnsp   |  relname  |     relhash
|           globalhash            |
              txcommand
----------+----------+----------+-------------------------------+-------+-----------+-----------+------------------
+----------------------------------+------------------
------------------------------------------------------------
       0 | testdb | omm      | 2021-04-14 07:00:46.32757+08  | 16393 | ledgernsp | usertable |
a41714001181a294 | 6b5624e039e8aee36bff3e8295c75b40 | insert into ledge
rnsp.usertable values(1, 'alex'), (2, 'bob'), (3, 'peter');
       1 | testdb | omm      | 2021-04-14 07:01:19.767799+08 | 16393 | ledgernsp | usertable |
b3a9ed0755131181 | 328b48c4370faed930937869783c23e0 | update ledgernsp.
usertable set name = 'bob2' where id = 2;
       2 | testdb | omm      | 2021-04-14 07:01:29.896148+08 | 16393 | ledgernsp | usertable |
0ae4b4e4ed2fcab5 | aa8f0a236357cac4e5bc1648a739f2ef | delete from ledge
rnsp.usertable where id = 3;
```

The result shows that the **omm** user has executed three consecutive DML commands: **INSERT**, **UPDATE**, and **DELETE**.

Step 2 Query records in the user history table.

```
gaussdb=# SELECT * FROM blockchain.ledgernsp_usertable_hist;
```

The result is as follows:

```
 rec_num |     hash_ins     |     hash_del     |             pre_hash
---------+------------------+------------------+----------------------------------
       0 | 1f2e543c580cb8c5 |                  | e1b664970d925d09caa295abd38d9b35
       1 | 8fcd74a8a6a4b484 |                  | dad3ed8939a141bf3682043891776b67
       2 | f51b4b1b12d0354b |                  | 53eb887fc7c4302402343c8914e43c69
       3 | 437761affbb7c605 | 8fcd74a8a6a4b484 | c2868c5b49550801d0dbbbaa77a83a10
       4 |                  | f51b4b1b12d0354b | 9c512619f6ffef38c098477933499fe3
(5 rows)
```

The result shows that the **omm** user has inserted three rows into the **ledgernsp.usertable** table, updated one row, and deleted one row. After these operations, two rows remain with hash values **1f2e543c580cb8c5** and **437761affbb7c605**.

Step 3 Query data in the user table and verify columns.

```
gaussdb=# SELECT *, hash_69dd43 FROM ledgernsp.usertable;
```

The result is as follows:

```
 id | name |       hash_69dd43
----+------+------------------
  1 | alex | 1f2e543c580cb8c5
  2 | bob2 | 437761affbb7c605
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The result shows that there are two records in the user table, matching the result in **Step 2**.

----End

7.5.3 Verifying Ledger Data Consistency

Prerequisites

The database is running properly, and a series of insert, delete, and update operations have been performed on the tamper-proof database to ensure that ledger operation records are generated in the query period.

Context

- Currently, the ledger database provides two verification APIs: ledger_hist_check(text, text) and ledger_gchain_check(text, text). When a common user calls a verification API, the API verifies only the tables accessible to the user.
- The pg_catalog.ledger_hist_check API is designed to verify tamper-proof user tables and user history tables. It can be called as follows:
  ```
  SELECT pg_catalog.ledger_hist_check(schema_name text,table_name text);
  ```
  If the verification is successful, the API returns **t**. Otherwise, it returns **f** along with the cause of failure.
- The pg_catalog.ledger_gchain_check API is designed to verify consistency across tamper-proof user tables, user history tables, and the global blockchain table. It can be called as follows:
  ```
  SELECT pg_catalog.ledger_gchain_check(schema_name text, table_name text);
  ```
  If the verification is successful, the API returns **t**. Otherwise, it returns **f** along with the cause of failure.

Procedure

Step 1 Check whether the tamper-proof user table **ledgernsp.usertable** is consistent with the corresponding user history table.

```
gaussdb=# SELECT pg_catalog.ledger_hist_check('ledgernsp', 'usertable');
```

The result is as follows:

```
 ledger_hist_check
-------------------
 t
(1 row)
```

The result confirms a one-to-one correspondence between entries in the tamper-proof user table and user history table.

Step 2 Check whether the tamper-proof user table **ledgernsp.usertable** matches records in the corresponding user history table and the global blockchain table.

```
gaussdb=# SELECT pg_catalog.ledger_gchain_check('ledgernsp', 'usertable');
```

The result is as follows:

```
 ledger_gchain_check
---------------------
 t
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The result shows that the records of **ledgernsp.usertable** in the preceding three tables are consistent and no tampering occurs.

----End

7.5.4 Archiving a Ledger Database

Prerequisites

- You are an audit administrator or a role that has the AUDITADMIN permission.
- The database is running properly, and a series of insert, delete, and update operations have been performed on the tamper-proof database to ensure that ledger operation records are generated in the query period.
- The audit file path has been correctly configured in **audit_directory**.

Context

- Currently, the ledger database provides two archiving APIs: ledger_hist_archive(text, text) and ledger_gchain_archive(text, text). Only audit administrators can call the ledger database APIs.
- The pg_catalog.ledger_hist_archive API is designed to archive the user history table of the current DN. It can be called as follows:
  ```
  SELECT pg_catalog.ledger_hist_archive(schema_name text,table_name text);
  ```
  If the archiving is successful, the API returns **t**. Otherwise, it returns **f** along with the cause of failure.
- The pg_catalog.ledger_gchain_archive API is designed to archive the global blockchain table of the current CN. It can be called as follows:
  ```
  SELECT pg_catalog.ledger_gchain_archive();
  ```
  If the archiving is successful, the API returns **t**. Otherwise, it returns **f** along with the cause of failure.

Procedure

Step 1 Run **EXECUTE DIRECT** to archive data on a DN.

```
gaussdb=# EXECUTE DIRECT ON (datanode1) 'select pg_catalog.ledger_hist_archive(''ledgernsp'', ''usertable'');';
```

The result is as follows:

```
 ledger_hist_archive
---------------------
 t
(1 row)
```

The user history table is archived as a record.

```
gaussdb=# EXECUTE DIRECT ON (datanode1) 'SELECT * FROM blockchain.ledgernsp_usertable_hist;';
 rec_num |     hash_ins     |     hash_del     |             pre_hash
---------+------------------+------------------+----------------------------------
       3 | e78e75b00d396899 | 8fcd74a8a6a4b484 | fd61cb772033da297d10c4e658e898d7
(1 row)
```

The result indicates that the user history table of datanode1 is successfully archived.

Step 2 Connect to a CN to archive its global blockchain table.

```
gaussdb=# SELECT pg_catalog.ledger_gchain_archive();
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The result is as follows:

```
 ledger_gchain_archive
-----------------------
 t
(1 row)
```

The global blockchain table is archived as *N* data records, one for each user table.

```
gaussdb=# SELECT * FROM gs_global_chain;
 blocknum |  dbname  | username |           starttime           | relid |  relnsp   |  relname  |     relhash
|           globalhash           | txcommand
----------+----------+----------+-------------------------------+-------+-----------+-----------+------------------
+----------------------------------+-----------
        1 | testdb | libc     | 2021-05-10 19:59:38.619472+08 | 16388 | ledgernsp | usertable |
57c101076694b415 | be82f98ee68b2bc4e375f69209345406 | Archived.
(1 row)
```

The result indicates that the global blockchain table of the current CN is successfully exported.

----End

7.5.5 Repairing a Ledger Database

Prerequisites

- You are an audit administrator or a role that has the AUDITADMIN permission.
- The database is running properly, and a series of insert, delete, and update operations have been performed on the tamper-proof database to ensure that ledger operation records are generated in the query period.

Context

- The current ledger database mechanism operates as follows: The global blockchain table is stored on CNs, with each CN maintaining independent data. User history tables, stored on DNs, record data changes in the DNs' tamper-proof tables. Consequently, data redistribution may lead to inconsistencies between tamper-proof tables and user history tables. In such scenarios, you need to call the ledger_hist_repair(text, text) API to restore user history tables on specified DNs. Then calling the history table verification API on these DNs will return **true**. When CNs are removed or repaired, data in the global blockchain table may be lost or become inconsistent with that in user history tables. In such scenarios, you need to call the ledger_gchain_repair(text, text) API to restore the global blockchain table across the entire cluster. Then calling the global blockchain table verification API will return **true**.
- The pg_catalog.ledger_hist_repair API is designed to restore user history tables. It can be called as follows:
  ```
  SELECT pg_catalog.ledger_hist_repair(schema_name text,table_name text);
  ```
  After a successful restoration, the API returns the incremental hash value generated during the restoration process for the user history table.

  Note: When performing flashback DROP on a user table, you can use this API to restore the names of the user table and user history table. For details, see **Restoring the Names of User Tables and User History Tables**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The pg_catalog.ledger_gchain_repair API is designed to restore the global blockchain table. It can be called as follows:

  ```
  SELECT pg_catalog.ledger_gchain_repair(schema_name text,table_name text);
  ```

  After a successful restoration, the API returns the total hash value generated during the restoration process for the specified table in the global blockchain table.

Restoring Data in a User History Table and the Global Blockchain Table

The **omm** user is used as an example. The procedure is as follows:

Step 1 Log in to the primary database node as the OS user **omm**.

Step 2 Run **EXECUTE DIRECT** to restore the user history table on a DN.

```
gaussdb=# EXECUTE DIRECT ON (datanode1) 'select pg_catalog.ledger_hist_repair(''ledgernsp'', ''usertable'');';
```

The result is as follows:

```
 ledger_hist_repair
--------------------
 84e8bfc3b974e9cf
(1 row)
```

The result indicates that the user history table of datanode1 is successfully restored, and the restoration leads to a hash increment of **84e8bfc3b974e9cf** for the user history table.

Step 3 Connect to a CN to restore its global blockchain table.

```
gaussdb=# SELECT pg_catalog.ledger_gchain_repair('ledgernsp', 'usertable');
```

The result is as follows:

```
 ledger_gchain_repair
----------------------
 a41714001181a294
(1 row)
```

The result indicates that the global blockchain table of the current cluster is successfully restored, and a restoration record is inserted into the current CN, with a hash value of **a41714001181a294**.

----End

Restoring the Names of User Tables and User History Tables

Before restoring the names of user tables and user history tables, ensure that flashback DROP has been enabled by setting the **enable_recyclebin** and **recyclebin_retention_time** parameters. For example:

- Perform flashback DROP on a user table. Call the ledger_hist_repair API to restore the names of the user table and corresponding user history table.

  ```
  -- Perform flashback DROP on the user table and call the ledger_hist_repair API to restore the name of the user history table.
  gaussdb=# CREATE TABLE ledgernsp.tab2(a int, b text);
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'rec_num' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  gaussdb=# DROP TABLE ledgernsp.tab2;
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
DROP TABLE
gaussdb=# SELECT rcyrelid, rcyname, rcyoriginname FROM gs_recyclebin;
 rcyrelid |           rcyname              |    rcyoriginname
----------+--------------------------------+---------------------
    32838 | BIN$39B523388046$55C8400==$0 | tab2
    32846 | BIN$39B52338804E$55C90E8==$0 | gs_hist_tab2_index
    32843 | BIN$39B52338804B$55C96A0==$0 | ledgernsp_tab2_hist
    32841 | BIN$39B523388049$55C9EE0==$0 | pg_toast_32838
(4 rows)
-- Perform flashback DROP on the user table.
gaussdb=# TIMECAPSULE TABLE ledgernsp.tab2 TO BEFORE DROP;
TimeCapsule Table
-- Call the ledger_hist_repair API to restore the name of the user history table.
gaussdb=# SELECT ledger_hist_repair('ledgernsp', 'tab2');
 ledger_hist_repair
--------------------
 0000000000000000
(1 row)

gaussdb=# \d+ ledgernsp.tab2;
                        Table "ledgernsp.tab2"
   Column    |  Type   | Modifiers | Storage  | Stats target | Description
-------------+---------+-----------+----------+--------------+-------------
 a           | integer |           | plain    |              |
 b           | text    |           | extended |              |
 hash_1d2d14 | hash16  |           | plain    |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=enhanced_toast
History table name: ledgernsp_tab2_hist

-- Perform flashback DROP on the user table and call the ledger_hist_repair API to restore the name
of the user table.
gaussdb=# CREATE TABLE ledgernsp.tab3(a int, b text);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'rec_num' as the distribution column by
default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# DROP TABLE ledgernsp.tab3;
DROP TABLE
gaussdb=# SELECT rcyrelid, rcyname, rcyoriginname FROM gs_recyclebin;
 rcyrelid |           rcyname              |    rcyoriginname
----------+--------------------------------+---------------------
    32952 | BIN$80B6233880B8$FECFF98==$0 | tab3
    32960 | BIN$80B6233880C0$FED0C98==$0 | gs_hist_tab3_index
    32957 | BIN$80B6233880BD$FED1250==$0 | ledgernsp_tab3_hist
    32955 | BIN$80B6233880BB$FED1A00==$0 | pg_toast_32952
(4 rows)
-- Perform flashback DROP on the user history table.
gaussdb=# TIMECAPSULE TABLE blockchain.ledgernsp_tab3_hist TO BEFORE DROP;
TimeCapsule Table
-- Obtain rcyname corresponding to the user table and call the ledger_hist_repair API to restore the
name of the user table.
gaussdb=# SELECT ledger_hist_repair('ledgernsp', 'BIN$80B6233880B8$FECFF98==$0');
 ledger_hist_repair
--------------------
 0000000000000000
(1 row)

gaussdb=# \d+ ledgernsp.tab3;
                        Table "ledgernsp.tab3"
   Column    |  Type   | Modifiers | Storage  | Stats target | Description
-------------+---------+-----------+----------+--------------+-------------
 a           | integer |           | plain    |              |
 b           | text    |           | extended |              |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 hash_7a0c87 | hash16  |          | plain    |             |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=enhanced_toast
History table name: ledgernsp_tab3_hist

-- Drop tables.
gaussdb=# DROP TABLE ledgernsp.tab2 PURGE;
DROP TABLE
gaussdb=# DROP TABLE ledgernsp.tab3 PURGE;
DROP TABLE
```

7.6 Label-based Mandatory Access Control

7.6.1 Defining Security Labels

Prerequisites

The database is running properly. To perform this operation, you must have the SYSADMIN permission or inherit the permissions of the built-in role gs_role_seclabel.

Context

The gs_security_label system catalog is provided to view security labels created in the system.

```
SELECT * FROM gs_security_label;
```

Procedure

Consider a real-world use case where a company classifies its information as public, confidential, and top secret, labeled as L1, L2, and L3, respectively. The stored information covers employees from Level 1 to Level 5, which you can define as five scopes: G1, G2, G3, G4, and G5. Based on this, you can begin defining security labels.

Step 1 Create a security label for public information in G2 and G4.

```
gaussdb=# CREATE SECURITY LABEL label1 'L1:G2,G4';
```

Step 2 Create a security label for confidential information from G2 to G4.

```
gaussdb=# CREATE SECURITY LABEL label2 'L2:G2-G4';
```

Step 3 Create a security label for top secret information from G1 to G5.

```
gaussdb=# CREATE SECURITY LABEL label3 'L3:G1-G5';
```

Step 4 Query the gs_security_label system catalog.

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

The result is as follows:

```
gaussdb=# SELECT * FROM gs_security_label;
 label_name | label_content
------------+---------------
 label1     | L1:G2,G4
 label2     | L2:G2-G4
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 label3     | L3:G1-G5
(3 rows)
```

Step 5 Drop security labels.

If you plan to run examples from other security label sections, complete them before performing this step. Otherwise, proceed directly with this step.

```
gaussdb=# DROP SECURITY LABEL label1;
gaussdb=# DROP SECURITY LABEL label2;
gaussdb=# DROP SECURITY LABEL label3;
```

----End

7.6.2 Applying Security Labels

Prerequisites

The database is running properly. To perform this operation, you must have the SYSADMIN permission or inherit the permissions of the built-in role gs_role_seclabel.

Context

- The pg_seclabel system catalog is provided to view security labels on tables or table columns.
  ```
  SELECT * FROM pg_seclabel;
  ```
- The pg_shseclabel system catalog is provided to view the security labels of users and roles.
  ```
  SELECT * FROM pg_shseclabel;
  ```
- The system view pg_seclabels is provided to view all security label records in the system.
  ```
  SELECT * FROM pg_seclabels;
  ```

Procedure

The database already has **user1**, **user2**, and a data table named **tbl**.

Step 1 Apply the security label **label1** to **user1**, indicating that the information level of **user1** is classified as public in G2 and G4.

```
gaussdb=# SECURITY LABEL ON USER user1 is 'label1';
```

Step 2 Apply the security label **label3** to **user2**, indicating that the information level of **user2** is classified as top secret from G1 to G5.

```
gaussdb=# SECURITY LABEL ON USER user2 is 'label3';
```

Step 3 Apply the security label **label2** to the **tbl** table, indicating that the information level of **tbl** is classified as secret from G2 to G4.

```
gaussdb=# SECURITY LABEL ON TABLE tbl is 'label2';
```

Step 4 Query the pg_seclabels system view.

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

The result is as follows:

```
gaussdb=# SELECT * FROM pg_seclabels;
 objoid | classoid | objsubid | objtype | objnamespace | objname | provider | label
--------+----------+----------+---------+--------------+---------+----------+--------
  16399 |     1259 |        0 | table   |         2200 | tbl     | maclabel | label2
```

```
 16391 |    1260 |       0 | role   |            | user1   | maclabel | label1
 16395 |    1260 |       0 | role   |            | user2   | maclabel | label3
(3 rows)
```

----End

7.6.3 Label-based Mandatory Access Control Checking

Prerequisites

- The database is running properly with mandatory access control (MAC) checking enabled (**enable_mac_check=on**).
- The user performing the operation has the necessary ACL permissions to access the table.

Context

Security label–based MAC policy rules are system-predefined and cannot be modified by users:

- INSERT policy: Data can be inserted only when the subject's security label level is lower than or equal to the object's security label level and the subject's security label range is a subset of the object's security label range.
- SELECT policy: Data can be queried only when the subject's security label level is higher than or equal to the object's security label level and the subject's security label range is a superset of the object's security label range.
- UPDATE and DELETE policies: Data can be modified or deleted only when the subject's security label level is equal to the object's security label level and the subject's security label range is equal to the object's security label range.
- If an object is unlabeled, MAC policies do not take effect on it.
- If a subject is unlabeled, it cannot access any labeled object.

Procedure

Step 1 Grant the SELECT and INSERT permissions on the **tbl** table to **user1** and **user2**.

```
gaussdb=# GRANT SELECT,INSERT ON tbl TO user1,user2;
```

Step 2 Log in to the database as **user1** and perform SELECT and INSERT operations. SELECT fails but INSERT succeeds, which complies with the MAC policy rules.

- In non-multi-primary mode:

```
gaussdb=> SELECT current_user;
 current_user
--------------
 user1
(1 row)
gaussdb=> SELECT * FROM tbl;
ERROR:  permission denied for relation tbl
DETAIL:  N/A
gaussdb=> INSERT INTO tbl VALUES (1);
INSERT 0 1
```

Step 3 Log in to the database as **user2** and perform SELECT and INSERT operations. SELECT succeeds but INSERT fails, which complies with the MAC policy rules.

- In non-multi-primary mode:

```
gaussdb=> SELECT current_user;
 current_user
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

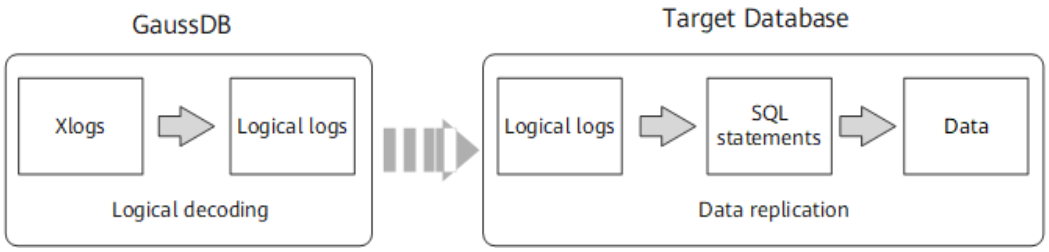
```
--------------
 user2
(1 row)
gaussdb=> SELECT * FROM tbl;
 fir
-----
   1
(1 row)
gaussdb=> INSERT INTO tbl VALUES (1);
ERROR:  permission denied for relation tbl -- This error occurs in non-multi-primary mode.
DETAIL:  N/A
```

----End

7.7 Logical Replication

Logical replication consists of logical decoding and data replication. Logical decoding extracts transaction-level logical logs. The service or database middleware parses the logs and replicates data. GaussDB supports periodically synchronizing data to heterogeneous databases using a data migration tool. However, it lacks real-time data replication; therefore, it cannot achieve real-time synchronization with heterogeneous databases. To this end, GaussDB provides the logical decoding feature to parse Xlogs and generate logical logs for the target database to parse in real time, implementing data replication. This feature lowers the requirements on the target database by supporting data synchronization between heterogeneous databases as well as between homogeneous but different databases. It allows data to be read and written during synchronization, reducing data synchronization latency. **Figure 7-3** shows the implementation. This section describes only logical decoding.

Figure 7-3 Logical replication



7.7.1 Logical Decoding (Non-Multi-Primary)

7.7.1.1 Overview

Description

Logical decoding provides basic transaction decoding capabilities for logical replication. GaussDB uses SQL functions for logical decoding. These functions are easy to call and require no additional tools. They provide straightforward APIs for interconnecting with external tools, eliminating the need for extra adaptation.

Logical logs are generated by transaction and can be output only after a transaction is committed. In addition, logical decoding is driven by users.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Therefore, to prevent Xlogs from being recycled by the system at the beginning of a transaction or prevent required transaction information from being recycled by VACUUM, GaussDB introduces logical replication slots to block Xlog recycling.

A logical replication slot represents a stream of changes that can be re-executed in other databases in the order they were generated in the original database. Each logical replication slot is maintained by the person who obtains the corresponding logical logs. If the database where the logical replication slot in streaming decoding resides does not have services, the replication slot is updated based on the log location of other databases. When processing an active transaction snapshot log, the database may update the active LSN-based logical replication slot based on the LSN of the current log. Similarly, when processing a virtual transaction log, it may update the active CSN-based logical replication slot based on the CSN of the current log.

NOTICE

In the current version, use decoding on directly connected DNs because the distributed strong-consistency decoding (CN decoding) feature is disabled by default. If required, contact Huawei technical support to enable the **enable-cn-decode** parameter after evaluation.

Prerequisites

- On the target CNs or DNs, SSL-related GUC parameters are set to **on**. This requirement applies because logical logs are extracted from CNs or DNs, necessitating SSL connections for logical replication.

  NOTE

  - You are advised to extract logical logs from DNs. Extracting logical logs from CNs involves numerous limitations and drawbacks. If extraction from CNs is required, contact Huawei technical support for assistance.
  - For security purposes, ensure that SSL connections are enabled.

- The GUC parameter **wal_level** is set to **logical**.

  NOTE

  If the GUC parameter **wal_level** is not set to **logical**, parallel decoding will skip non-logical log segments and will not output decoding logs; serial decoding will trigger an error.

- The GUC parameter **max_replication_slots** is set to a value greater than or equal to the total number of physical streaming replication slots, backup slots, and logical replication slots required by each node.

  NOTE

  - Each logical replication slot decodes only one database. To decode multiple databases, create multiple logical replication slots.
  - During multi-channel logical replication and synchronization, the source database must create an independent logical replication slot for each logical replication link.
  - A single CN can run a maximum of 10 distributed strong-consistency decoding tasks at the same time. A single DN can run a maximum of 20 decoding tasks at the same time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- To directly use SQL functions for logical decoding (see **Logical Decoding by SQL Functions**), you need to connect to the database through a DN port. If a CN port is used instead, SQL functions must be executed via **EXECUTE DIRECT ON (***datanode_name***)** *'statement'*.
- Only the initial user and users with the REPLICATION permission are authorized to perform logical replication. Database administrators are also granted this authorization when separation of duties is disabled. However, when separation of duties is enabled, database administrators are prohibited from carrying out logical replication operations.

Precautions

- DML decoding is not supported for data page replication.
- Logical replication does not support online cluster scale-out. Before online scale-out, you must delete existing logical replication slots. After the scale-out is completed, create them again.
- If a DDL operation fails due to unsuccessful DDL reverse parsing, you can retry the operation. In extreme cases where the retry also fails, set the GUC parameter **enable_logical_replication_ddl** to **OFF** to disable DDL reverse parsing and prioritize service continuity.
- During parallel decoding, if **decode-style** is **'b'** or **'t'**, the system discards any decoded data equal to or exceeding 1,023 MB per tuple (the combined size of original data and new data in an inserted or updated tuple) and logs the event. If **decode-style** is **'j'**, it discards decoded data exceeding 500 MB per tuple and logs the event. To prevent data loss, ensure that no single tuple exceeds 500 MB in size.
- If **wal_level** is set to **logical** and **REPLICA IDENTITY** is set to **FULL**, updating table tuples larger than 1 GB will fail.
- GaussDB can decode the following data types: INTEGER, BIGINT, SMALLINT, TINYINT, SERIAL, SMALLSERIAL, BIGSERIAL, FLOAT, DOUBLE PRECISION, BOOLEAN, BIT(n), BIT VARYING(n), DATE, DATEA, TIME[WITHOUT TIME ZONE], TIMESTAMP[WITHOUT TIME ZONE], CHAR(n), VARCHAR(n), TEXT, and CLOB (decoded into the text format).
- In ORA-compatible mode, both empty strings (with a length of 0) and NULL values can be inserted into columns of the BLOB/CLOB type. In the binary output of decoding, they can be differentiated based on the length of the column values: **0xFFFFFFFF** represents NULL, while **0** represents a zero-length string. For example, if inserting the column value returned by the empty_blob() function generates a non-NULL empty BLOB, the decoding process outputs a zero-length string; if **''** is inserted as the column value, the decoding process outputs NULL.
- In a non-M-compatible database containing floating-point data, or an M-compatible database containing floating-point data without scale, **extra_float_digits** (which controls the display precision of decoded results) is set to **3**. (For details, see the description of the GUC parameter **float_shortest_precision**.)
- DDL operations related to the **PUBLIC** schema cannot be decoded.
- Logical decoding output does not include session parameter settings or other status information (such as **check_function_bodies** or **b_format_version**). If statement execution results depend on such status information, ensure that

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

session parameters are consistent between the source and destination systems during replay on the destination system. Otherwise, errors or data inconsistencies may occur.

- Logical decoding is supported for DML and DDL operations in M-compatible databases. The supported data types are as follows:
  - Integer: TINYINT(M), SMALLINT(M), MEDIUMINT(M), BIGINT(M), INT(M)/INTEGER, and BOOL/BOOLEAN.
  - Floating-point: FLOAT(M,D) and DOUBLE(M,D).
  - Fixed-point: DECIMAL(M,D) and NUMERIC(M,N).
  - BIT: BIT.
  - Binary string: CHAR(N), VARCHAR(N), BINARY, and VARBINARY.
  - Text: TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT.
  - Date: DATE, TIME, DATETIME, TIMESTAMP, and YEAR.
  - LOB: TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB.
  - Data type attribute: UNSIGNED and ZEROFILL.
- In an M-compatible database, logical decoding is subject to the following special limitations:
  - In the CREATE TABLE, ALTER TABLE, and DROP TABLE syntaxes, options like **[partition_options]**, **ENGINE**, **ROW_FORMAT**, **algorithm_option**, and **lock_option** have no effect. The database neither decodes nor outputs them.
  - In the ALTER SCHEMA, CREATE SCHEMA, DROP SCHEMA, ALTER DATABASE, CREATE DATABASE, and DROP DATABASE syntaxes, ALTER DATABASE, CREATE DATABASE, and DROP DATABASE are decoded as ALTER SCHEMA, CREATE SCHEMA, and DROP SCHEMA because they are equivalent to schemas in M-compatible mode.
  - Different character setting syntaxes, such as CHARACTER SET, CHAR SET, and CHARSET, are decoded as CHARACTER SET.
  - For the ALTER TABLE tbl_name DROP {INDEX | KEY} index_name syntax, the logical decoding result is DROP INDEX. For the ALTER TABLE tbl_name DROP {primary key | {index | key} index_name} syntax for deleting a primary key, the logical decoding result is ALTER TABLE tbl_name DROP CONSTRAINT index_name.
  - For the ALTER TABLE tbl_name ADD INDEX syntax, the logical decoding result is CREATE INDEX.
  - During DML statement decoding, if the inserted data is of the time type (time, timestamp, or datetime) with precision, the maximum precision 6 is used for decoding.
  - For binary data types (BINARY, VARBINARY, TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB), input data containing '\0' will be truncated. Binary output is generated only during parallel decoding when the **decode-style** parameter is set to **'b'**. Truncation occurs during serial decoding, function decoding, parallel decoding, or when **decode-style** is set to **'j'** (JSON format) or **'t'** (text format).
- REPLACE is a DML syntax and will be decoded as an actual data operation, such as INSERT or DELETE+INSERT (in case of a primary key or unique key conflict).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- COPY/LOAD is a DML syntax and will be decoded as actual data operations, such as multiple INSERT operations.
- A logical replication slot name must be under 64 characters. When you call SQL functions to create or use a replication slot, the name can include only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). Note that a single period (.) or two periods (..) alone cannot serve as a replication slot name. When you call JDBC APIs to create or use a replication slot, the name can include only lowercase letters, digits, and underscores (_). Although you can enter uppercase letters, the database automatically converts them to lowercase—for example, **MSLOT** becomes **mslot**.
- To decode multiple databases, you need to create a streaming replication slot in each database and start decoding. Logs need to be scanned for decoding of each database.
- Forcible switchover is not supported during logical decoding. After forcible switchover, you need to export all data again.
- During decoding on the standby node, switchover or failover can increase decoded data, entailing the need for manual filtering. When the quorum protocol is used, switchover and failover should be performed on the standby node that is to be promoted to primary, and logs must be synchronized from the primary node to the standby node.
- If you need to delete a database during decoding on the standby node, ensure that the database of the corresponding logical replication slot is deleted by the primary node after data migration is complete. Currently, it is not necessary to delete the logical replication slot on a database before the database is deleted. As a result, the database may be deleted from the primary node before the decoding on the standby node is complete. In this case, when the standby node replays to the time when the database is deleted, the standby node exits decoding even if it has not completed all decoding tasks in the current database. If the standby node reconnects to the database, decoding on the standby node cannot be started because the database has been deleted. As a result, decoding tasks are damaged.
- During decoding on the standby node, function decoder threads and streaming serial decoder threads may be terminated due to conflicts with replay on the standby node. If a big transaction needs to be decoded, adjust the GUC parameters **max_standby_streaming_delay**, **standby_max_query_time**, and **standby_force_recycle_ratio** appropriately based on the estimated decoding duration.
- Replication slots can be created or deleted only for CNs and primary DNs. If the replication slot to be deleted is the last one, alarms "replicationSlotMinLSN is INVALID_WAL_REC_PTR!!!" and "replicationSlotMaxLSN is INVALID_WAL_REC_PTR!!!" are generated after the deletion is complete.
- If a replication slot is deleted from the primary node while a standby node is experiencing severe Xlog replay lag, the corresponding slot on the standby node may not be removed promptly. Subsequent decoding attempts using this slot on the standby node will fail when the deletion operation is replayed. Before using a replication slot on a standby node for decoding, it is advisable to verify that its **restart_lsn** has a match on the primary node. If no match is found, the slot is considered residual.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Primary and standby DNs or different standby DNs must not share the same replication slot for decoding. Otherwise, data inconsistency or other errors may occur.
- After database restart due to faults or logical replication process restart, duplicate decoded data may be generated. Manual filtering is required to remove duplicate entries.
- If the computer kernel is faulty, garbled characters may be displayed during decoding, which need to be manually or automatically filtered out.
- Ensure that no long-running transaction is started during the creation of a logical replication slot. Starting a long-running transaction can block the creation of the logical replication slot.
- When both the primary and standby nodes decode data at the same time, if the primary node is faulty or a switchover is performed, a node may encounter more than 20 replication links (because the replication links of the original primary node are transferred to the new primary node).
- Decoding DML operations on global temporary tables is not supported.
- Decoding DML operations on local temporary tables is not supported.
- For the SELECT INTO statement, only the DDL operations for creating the target table are decoded; the DML operations for inserting data will not be decoded.
- To parse the UPDATE and DELETE statements on an Astore table, you must configure the **REPLICA IDENTITY** attribute for the table. If the table lacks a primary key, set **REPLICA IDENTITY** to **FULL**. For detailed configuration, refer to **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }** in "SQL Reference > SQL Syntax > A > ALTER TABLE" in *Reference*.
- Do not operate on a logical replication slot from other nodes while it is in use. To delete a replication slot, ensure that it has stopped decoding.
- Considering that the target database may require the system status information of the source database, logical decoding automatically filters only logical logs of system catalogs whose OIDs are less than 16384 in pg_catalog and pg_toast schemas. If the target database does not need to copy the content of other related system catalogs, the related system catalogs need to be filtered during logical log replay.
- To create a primary key index that includes system columns when logical replication is enabled, you must set the **REPLICA IDENTITY** attribute of the table to **FULL** or include **USING INDEX** to specify a unique, non-local, non-deferrable index that excludes system columns and includes only columns marked **NOT NULL**.
- When logical replication is enabled, you are advised to define PRIMARY KEY or UNIQUE constraints for user tables to ensure deterministic replay. Otherwise, SQL statement replay may modify multiple rows, compromising data accuracy.
- If a replication table exists prior to scale-in or upgrade, the **logical_repl_node** attribute of the replication table must be manually configured or reset to the default value. For detailed configuration, refer to the descriptions of the **storage_parameter** parameter and **logical_repl_node** attribute in "SQL Reference > SQL Syntax > ALTER TABLE" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If a transaction contains excessive sub-transactions, numerous files may be flushed to disk. To terminate decoding, you need to manually execute the SQL function pg_terminate_backend(*WALSender thread ID for logical decoding*). In addition, the exit delay increases by approximately 1 minute per 300,000 sub-transactions. Therefore, when logical decoding is enabled, a WARNING event is logged if the number of sub-transactions in a transaction reaches 50,000.
- During logical decoding of a big transaction, the system temporarily writes intermediate decoding results to the data disk. The ratio of Xlog size to flushed-to-disk file size is less than 1:1.5, meaning that for every 1 GB of Xlogs written by the big transaction, no more than 1.5 GB of files may be flushed to disk, resulting in high disk usage. Therefore, it is advisable to reserve sufficient disk space on nodes to ensure that the safe threshold is maintained even during peak decoding periods.
- When a logical replication slot is inactive, the GUC parameter **enable_xlog_prune** is set to **on**, **enable_logicalrepl_xlog_prune** is set to **on**, and **max_size_for_xlog_retention** is set to a non-zero value, the number of retained log segments caused by the backup slot or logical replication slot exceeds **wal_keep_segments**, and no other replication slots cause more log segments to be retained, the logical replication slot will be forcibly invalidated and its **restart_lsn** will be set to **7FFFFFFF/FFFFFFFF** under either of the following conditions: **max_size_for_xlog_retention** is greater than 0 and the number of retained log segments (each 16 MB in size) caused by the current logical replication slot exceeds **max_size_for_xlog_retention**, or **max_size_for_xlog_retention** is less than 0 and disk usage reaches **– max_size_for_xlog_retention/100**. Logical replication slots in this state do not participate in the recycling of blocked logs or historical system catalogs, but the limitation on the maximum number of replication slots still takes effect. In this case, you need to manually delete them.
- After the standby node starts decoding and sends an instruction of updating the replication slot number to the primary node, the standby node occupies a corresponding logical replication slot (identified as an active state) on the primary node. Before that, the corresponding logical replication slot on the primary node is inactive. In this state, if the condition for forcibly invalidating the logical replication slot is met, the logical replication slot is marked as invalid (that is, **restart_lsn** is set to **7FFFFFFF/FFFFFFFF**). As a result, the standby node cannot update the replication slot on the primary node. In addition, after the standby node replays the logs indicating that the replication slot is invalid, the standby node of the current replication slot cannot be reconnected if decoding is disconnected.
- When using standby nodes for decoding, it is advisable to consistently start decoding tasks to on the same designated standby node. If a logical replication slot on a standby node is forcibly invalidated, attempting to start decoding tasks on that node will result in an error.
- During serial logical decoding on a standby node, replay lag may cause a failure to obtain the decoding snapshot and trigger an error. In such a case, you are advised to retry decoding once the replay on the standby node is complete.
- Inactive logical replication slots will block WAL recycling and the cleanup of historical system catalog tuples. This causes disk logs to accumulate and degrades system catalog scan performance. Therefore, unused logical replication slots should be cleaned up promptly. Notably, if a logical

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

replication slot is created by connecting to a DN's extended IP address during the observation period before an upgrade is submitted, you must manually clean up the replication slot prior to rolling back the upgrade. Otherwise, once the DN's extended IP address is rolled back, direct connection to the DN will no longer be possible.

- Distributed strong-consistency logical decoding (connected to CNs) is supported only with GTM-Lite distributed deployment and streaming decoding. It is not supported in scenarios where CNs are connected to standby DNs for decoding or in cases involving SQL logical decoding functions, online scale-out, or global indexes.
- For distributed strong-consistency logical decoding (connected to CNs), services take charge of switchovers to maintain high CN availability.
- CSN-based logical replication slots on CNs serve only as placeholders. Their slot numbers do not update along with the progress of logical decoding, and they do not block log recycling.
- When logical replication slots are created by connecting to CNs via protocols, only CSN-based replication slots are supported. When logical replication slots are created by connecting to DNs via protocols, only LSN-based replication slots are supported.
- For distributed decoding, if an error occurs or the decoding client is manually stopped, wait for 15 seconds before attempting decoding again. If a replication slot is occupied, call the SQL function pg_terminate_backend(*ID of the thread that occupies the replication slot*) to manually release the replication slot.
- If a replication slot creation attempt fails on a CN, the replication slot must be deleted on the same CN before a new creation can be attempted.
- If the replication slot you wish to delete from a CN is LSN-based, it will be deleted only from the current node, leaving same-named slots on other nodes unaffected (if any). Otherwise, as long as other nodes have same-named CSN-based logical replication slots, the deletion will not trigger an error even if the replication slot is not found on some nodes; instead, all same-named slots will be deleted from all nodes. However, if the replication slot is not found on any node, an error will be reported.
- When a CSN-based logical replication slot is created on a CN, if same-named LSN-based logical replication slots exist on some nodes, those residual slots must be deleted first. Otherwise, the CSN-based logical replication slot will be created on all CNs (excluding the current CN) and primary DNs that do not already have a same-named replication slot.
- If an LSN-based logical replication slot exists on the current CN and a same-named CSN-based logical replication slot exists on other nodes, deleting the replication slot on the current CN will only remove the local LSN-based logical replication slot. After the deletion is complete, repeat the deletion operation to remove the same-named replication slot from other nodes.
- When the JSON format is used for decoding, the data column cannot contain special characters (such as the null character '\0'). Otherwise, the content in the decoding output column will be truncated.
- Decoding DML operations on unlogged tables is not supported.
- During backup and restoration, all logical replication slots will be cleaned up upon completion of instance restoration. If needed, rebuild these slots. Note

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

that backup and restoration require stopping all decoding tasks; otherwise, logical replication slots may fail to be cleaned up.

- After cluster-level database restoration, the data dictionary replication slot of the database becomes unusable. In this case, you need to re-execute instance-level logical decoding to baseline the data dictionary. (This will invalidate data dictionary replication slots for other databases on the instance, entailing the need to rebuild these slots.) Evaluate the risk of slot invalidation for other databases before baselining.
- Replication slots must be deleted before their database is dropped. Once the database hosting logical replication slots is dropped, those slots will become unavailable and must be deleted manually. Failure to do so will block WAL recycling.
- When a transaction generates a large number of sub-transactions that need to be flushed to disk, the number of open file handles may exceed the upper limit. In this case, set the GUC parameter **max_files_per_process** to a value greater than twice the upper limit of sub-transactions.
- GSIs are not supported. Additionally, DML statements that modify distribution keys cannot be decoded.
- The standby cluster in a DR system does not support logical decoding via SQL system functions or tools.
- The ledger database functionality is not supported. In the current version, if the database used for decoding tasks contains DML operations related to a ledger database, the decoding results will include a hash column, causing replay to fail.
- In scale-out scenarios, if logical replication slots exist in the cluster, the scale-out will fail. Therefore, you must delete all existing logical replication slots before initiating scale-out.
- Information constraint options (such as **not enforced**) in PRIMARY KEY, FOREIGN KEY, and UNIQUE constraints are not supported. Information constraints in DDL statements are not decoded, while other parts of the DDL statements are decoded as usual. For example, **CREATE TABLE test(a int primary key not enforced);** will be decoded as **CREATE TABLE test(a int);**.
- When the CREATE TABLE syntax is used to create a table, the **MURMURHASH** option is not allowed. If the **MURMURHASH** option is included, the DDL statement cannot be decoded.
- Only logical WALs (**wal_level=logical**) can be decoded. For non-logical logs, serial decoding (including function decoding) does not output relevant values or types, and parallel decoding does not output logical logs.
- Logical decoding supports the following types of DML operations on Xlogs:
  - Astore: INSERT, DELETE, UPDATE, and MULTI-INSERT.
  - Ustore: INSERT, DELETE, UPDATE, and MULTI-INSERT.
- Compared with the scenario where **wal_level** is set to **hot_standby**, the Xlog data volume increases when **wal_level** is set to **logical**. For example, in the TPC-C scenario, the Astore Xlog bloat rate is about 11%, and the Ustore Xlog bloat rate is about 110%.
- When decoding is performed in a case-insensitive database, the table names or usernames specified in the decoding option whitelist (**white-table-list**) or blacklist (**exclude-users**) must be in lowercase, regardless of their original case.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Function decoding does not support the 0x character.
- In scenarios where decoding is performed on directly connected DNs, if the decoding process encounters an uncommitted transaction, it must wait for the transaction to be committed and acknowledged. If the wait time exceeds **transaction_sync_timeout**, the decoding process will be terminated due to a timeout.
- If the ILM compression syntax specifies a policy ID, the source and target instances may use different policy IDs for the table because of the advanced compression mechanism.

```
gaussdb=# ALTER TABLE tbl_ilm ILM ENABLE POLICY p1; -- p1 is the policy ID of the source database.
gaussdb=# ALTER TABLE tbl_ilm ILM DISABLE POLICY p1; -- p1 is the policy ID of the source database.
gaussdb=# ALTER TABLE tbl_ilm ILM DELETE POLICY p1; -- p1 is the policy ID of the source database.
```

- Logical replication slots prevent the system catalog versions corresponding to unparsed Xlogs from being cleared prematurely. If a service involves a large number of DDL operations and the relevant logical replication slot is inactive or the client consumes data slowly, excessive historical versions of system catalogs may accumulate. Because of its data page management limitations, GaussDB cannot store two records on the same page if their XID difference exceeds 2^32. When excessive system catalog versions exist, service operations such as DDL and DML may fail to execute, resulting in error code GAUSS-21297.

  Solution:

  – Promptly clean up unused logical replication slots.
  – If a logical replication slot updates too slowly, contact technical support for assistance.
- Parallel logical decoding tasks consume CPU resources. Excessive parallel decoding tasks may lead to resource contention, affecting online services. According to TPC-C testing on an x86 system with 128-core CPUs and 1,024 GB of memory, each parallel decoding task (with a DOP of 4) reduces online operation performance by approximately 15%.
- Insufficient system resources or environmental fluctuations may cause decoder thread creation failures or startup timeouts, leading to parallel logical decoding task failures with the error: "Wait for worker ready failed." If the issue occurs occasionally, retry logical decoding on the client as a temporary workaround. If the issue recurs frequently, reduce the overall cluster load to ensure sufficient resources for decoding tasks.
- If database object names (such as table, column, and index names) are system keywords (for example, FLOAT and ENTITYESCAPING), the decoding output may not correctly escape or enclose these keywords in backticks (`) or double quotes ("). Therefore, when applying the decoded output to the target database, you must follow its syntax rules and manually add or adjust the backticks (`) or double quotes (") around keywords as needed to avoid syntax parsing errors.
- For tables with non-replication distribution, DN decoding only parses WALs from the current node.

SQL Function Decoding Performance

1. In the Benchmarksql-5.0 with 100 warehouses, when pg_logical_slot_get_changes is used:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - If 4,000 lines of data (about 5 MB to 10 MB of logs) are decoded at a time, the decoding performance ranges from 0.3 MB/s to 0.5 MB/s.
  - If 32,000 lines of data (about 40 MB to 80 MB of logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
  - If 256,000 lines of data (about 320 MB to 640 MB of logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
  - Further increasing the data volume decoded at a time will not noticeably improve decoding performance.

  If pg_logical_slot_peek_changes and pg_replication_slot_advance are used, the decoding performance is 30% to 50% lower than that when pg_logical_slot_get_changes is used.

2. In the Benchmarksql-5.0 with 100 warehouses, when pg_logical_get_area_changes is used:
  - If 4,000 lines of data (about 5 MB to 10 MB of logs) are decoded at a time, the decoding performance ranges from 0.3 MB/s to 0.5 MB/s.
  - If 32,000 lines of data (about 40 MB to 80 MB of logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
  - If 256,000 lines of data (about 320 MB to 640 MB of logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
  - Further increasing the data volume decoded at a time will not noticeably improve decoding performance.
3. In the Benchmarksql-5.0 with 100 warehouses, when pg_logical_slot_get_binary_changes is used:
  - If 4,000 lines of data (about 5 MB to 10 MB of logs) are decoded at a time, the decoding performance ranges from 0.3 MB/s to 0.5 MB/s.
  - If 32,000 lines of data (about 40 MB to 80 MB of logs) are decoded at a time, the decoding performance ranges from 2 MB/s to 3 MB/s.
  - If 256,000 lines of data (about 320 MB to 640 MB of logs) are decoded at a time, the decoding performance ranges from 2 MB/s to 3 MB/s.
  - Further increasing the data volume decoded at a time will not noticeably improve decoding performance.

  If pg_logical_slot_peek_binary_changes and pg_replication_slot_advance are used, the decoding performance is 30% to 50% lower than that when pg_logical_slot_get_binary_changes is used.

Streaming Decoding Performance

The parallel decoding performance (measured by Xlog consumption) is no less than 100 MB/s when a DN is connected or no less than 80 MB/s when a CN is connected in this standard setup: 16-core CPU, 128 GB of memory, network bandwidth > 200 MB/s, 10 to 100 columns in a table, 0.1 KB to 1 KB of data per row, INSERT-dominant DML operations, no transactions flushed to disk (that is, fewer than 4,096 statements per transaction), **parallel-decode-num=8**, **decode-style='t'**, and **sending-batch** enabled. To ensure that the decoding performance meets the requirements and to minimize the impact on services, you are advised to set up only one parallel decoding connection on a standby node to ensure that the CPU, memory, and bandwidth resources are sufficient.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.7.1.2 Logical Decoding Options

Logical decoding options are designed to restrict or enhance functionalities for the current logical decoding process. For example, they allow you to specify whether to include transaction IDs in decoding results or whether to ignore empty transactions during decoding. In the case of SQL function decoding, refer to **'options_name'** and **'options_value'** (optional input parameters for the pg_logical_slot_peek_changes function) in "SQL Reference > Functions and Operators > System Administration Functions > Logical Replication Functions" in *Reference* for detailed configuration methods. In the case of JDBC streaming decoding, refer to the usage of the withSlotOption function in the example code provided in **Logical Replication (Non-Multi-Primary)**.

General Options

General options can be configured for both serial decoding and parallel decoding, although they may not always take effect. See their respective descriptions for details.

- **include-xids**:

  Specifies whether the decoded **data** column includes XIDs.

  Value type: Boolean.

  Value range:

  - **false**: The decoded **data** column does not include XIDs.
  - **true**: The decoded **data** column includes XIDs.

  Default value: **true**.

- **skip-empty-xacts**:

  Specifies whether to ignore empty transactions during decoding.

  Value type: Boolean.

  Value range:

  - **false**: Empty transactions are not ignored during decoding.
  - **true**: Empty transactions are ignored during decoding.

  Default value: **false**.

- **include-timestamp**:

  Specifies whether the decoded information includes commit timestamps.

  Value type: Boolean.

  Value range:

  - **false**: The decoded information does not include commit timestamps.
  - **true**: The decoded information includes commit timestamps.

  Default value: **false** in parallel decoding scenarios or **true** in SQL function decoding and serial decoding scenarios.

- **only-local**:

  Specifies whether to decode only local logs.

  Value type: Boolean.

  Value range:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **false**: Both local and non-local logs are decoded.
  - **true**: Only local logs are decoded (that is, **RepOriginId=0**, which is explained under the "Database GUC Parameters > GUC Parameters > Dual-Cluster Replication Parameters" section in *Reference*).

  Default value: **true**.

- **white-table-list**:

  Specifies the whitelist of schemas and tables to be decoded.

  Value range: a string of whitelisted table names, separated by commas (,). You can use an asterisk (*) for fuzzy matching of all. Separate schema names and table names with periods (.), and avoid including any whitespace. For example:

  ```
  SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list',
  'public.t1,public.t2,*.t3,my_schema.*');
  ```

  NOTE

  - When an ALTER VIEW statement is executed on an invalid view, the statement is recorded for logical decoding. In this case, the **white-table-list** parameter does not take effect. That is, the ALTER VIEW statement for the invalid view will be decoded, whether the table queried by the view definition statement is included in the whitelist.
  - For details about how to recompile an invalid view, see **Recompilation upon Invalidation**.
  - If tables in the whitelist remain idle for an extended period, the **restart_lsn** of the replication slot will not advance, leading to WAL accumulation. To resolve this issue, you can periodically call the system function gs_add_logical_decoding_position_xlog().

- **max-txn-in-memory**:

  Defines memory control. If a single transaction occupies more memory than this limit, it is flushed to disk.

  Unit: MB.

  Value range for serial decoding: integers from 0 to 100. If set to the default value of **0**, this memory control is disabled.

  Value range for parallel decoding: 0 to 25% of **max_process_memory**. The default value is calculated as **max_process_memory/4/1024**, where **1024** converts the unit from KB to MB. If set to **0**, this memory control is disabled.

- **max-reorderbuffer-in-memory**:

  Defines memory control. If the total memory (including cache) required by transactions being concatenated in the sender thread exceeds this limit, the currently decoded transaction is flushed to disk. In parallel decoding mode, this parameter has been deprecated and no longer takes effect.

  Unit: GB.

  Value range for serial decoding: integers from 0 to 100. If set to the default value of **0**, this memory control is disabled.

  Value range for parallel decoding: 0 to 50% of **max_process_memory**. The default value is calculated as **max_process_memory/2/1048576**, where **1048576** converts the unit from KB to GB. If set to **0**, this memory control is disabled.

- **desc-memory-limit**:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Defines memory control. If the total memory required by the table metadata maintained by a logical decoding task exceeds this limit, some table metadata will be evicted to maintain the limit.

Value type: integer.

Unit: MB.

Value range: 10–1048576.

Default value: **200**.

NOTE

After the kernel is upgraded from a version earlier than 505.2.1.SPC0900 to the current version, performance may degrade in certain scenarios. Based on best practices, the following configuration is recommended:

1. Set **desc-memory-limit** to 100 MB for scenarios involving 1,000 hot tables—each with an average of 100 columns and 50 partitions—where a new version is generated every 10 minutes (due to operations such as executing DDL modifications to table definitions or performing AUTOVACUUM to modify **reloptions** in the pg_class system catalog).
2. If any of the preceding indicators increase, adjust the configuration proportionally. For example, if there are 2000 hot tables with an average of 200 columns each, and a new version is generated every five minutes, adjust **desc-memory-limit** to 800 MB (100 × 2 × 2 × 2).

- **include-user**:

  Specifies whether to include the transaction username in the BEGIN logical log of a transaction. The transaction username refers to an authorized user (that is, the logged-in user executing the session associated with the transaction). This username remains unchanged throughout the execution of the transaction.

  Value type: Boolean.

  Value range:

  – **false**: The transaction username is excluded from the BEGIN logical log of the transaction.

  – **true**: The transaction username is included in the BEGIN logical log of the transaction.

  Default value: **false**.

- **exclude-userids**:

  Lists the OIDs of blacklisted users. This parameter is configurable only for decoding tasks on directly connected DNs and is not supported for distributed strong-consistency decoding tasks on CNs.

  Value type: string.

  Value range: OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not verify the existence of the specified OIDs.

- **exclude-users**:

  Lists the names of blacklisted users.

  Value type: string.

  Value range: names of blacklisted users. Multiple names are separated by commas (,). The system does not verify the existence of the specified names.

- **dynamic-resolution**:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies whether to dynamically resolve the names of blacklisted users. When decoding an Xlog, if a user did not exist at the time the Xlog was written, the system considers the user non-existent.

Value type: Boolean.

Value range:

- **false**: When finding that the relevant user is not present in the blacklist (**exclude-users**), the system reports an error and terminates logical decoding. If the user is present, it is excluded by the blacklist mechanism.
- **true**: When finding that the relevant user is not present in the blacklist (**exclude-users**), the system proceeds with decoding without reporting an error. If the user is present, it is excluded by the blacklist mechanism.

Default value: **true**.

- **standby-connection**:

  (Configurable only for streaming decoding) Specifies whether to restrict decoding to standby nodes only. Since CNs lack the standby role, this parameter is supported only when DNs are connected.

  Value type: Boolean.

  Value range:

  - **true**: Decoding is restricted to standby nodes only. If a connection is attempted to the primary node, the system reports an error and exits.
  - **false**: No restrictions apply. Decoding can be performed on both primary and standby nodes.

  Default value: **false**.

  NOTE

  If the primary node experiences high resource usage and services are insensitive to real-time incremental data synchronization, you are advised to perform decoding on standby nodes. If services require real-time synchronization and the primary node is under low service pressure, decoding on the primary node is recommended.

- **sender-timeout**:

  (Configurable only for streaming decoding) Specifies the heartbeat timeout threshold between GaussDB and clients. If no message is received from a client within this period, logical decoding tasks will proactively stop and disconnect from the client.

  Value type: integer.

  Unit: ms.

  Value range: 0–2147483647.

  Default value: depending on the GUC parameter **logical_sender_timeout**. If set to **0**, logical decoding tasks will not proactively disconnect from clients. If set to a small value (for example, 1 ms), decoding tasks may be interrupted prematurely.

  NOTE

  sender-timeout specifies heartbeat detection threshold. Once the detection times out, the system automatically retries. If retries fail, the system proactively closes the connection and terminates the decoding task.

- **change-log-max-len**:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the upper length limit for caching logical logs. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding. If the length of a single decoding result exceeds this limit, the originally allocated memory will be destroyed, and a new 1024-byte memory block will be allocated to cache the result. Setting a high limit increases memory usage, whereas setting a low limit may cause frequent memory allocation and deallocation. Avoid setting a value below **1024**.

Unit: bytes.

Value range: 1–65535.

Default value: **4096**.

- **max-decode-to-sender-cache-num**:

  Specifies the maximum number of parallel-decoded logs that can be cached locally. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding. When the number of locally cached logs is insufficient, additional logs will be fetched from the global cache.

  Value range: 1–65535.

  Default value: **8192**.

  When a decoding task starts, the maximum number of cached parallel-decoded logs is controlled by the **max-queue-cache-num** parameter. Compatibility note: If set below **8192**, **max-decode-to-sender-cache-num** will be automatically adjusted when a decoding task starts, with a corresponding message recorded in system logs.

- **enable-heartbeat**:

  (Configurable only for streaming decoding) Specifies whether to generate heartbeat logs.

  Value type: Boolean.

  Value range:

  – **true**: Heartbeat logs are generated.

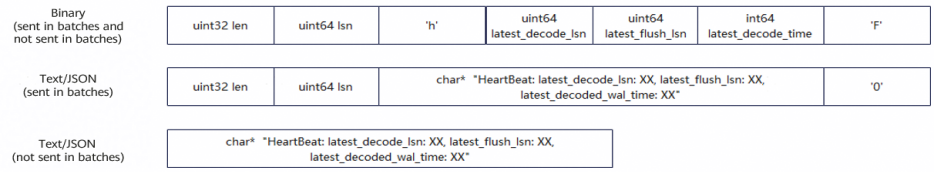
  – **false**: Heartbeat logs are not generated.

  Default value: **false**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

Assuming the heartbeat log option is enabled, the following explains how heartbeat logs are parsed in parallel decoding scenarios: In binary format, the log message starts with character 'h' (indicating a heartbeat log). It then includes the following content: an 8-byte uint64 value (during decoding on directly connected DNs, it is an LSN indicating the end position of the WAL read when the heartbeat logical log was sent; in distributed strong-consistency decoding scenarios, it represents the decoded log's transaction CSN that had been sent when the heartbeat logical log was sent); another 8-byte uint64 value (during decoding on directly connected DNs, it is an LSN indicating the position of the WAL that had been flushed to disk when the heartbeat logical log was sent; in distributed strong-consistency decoding scenarios, it represents the CSN to be assigned to the next committed transaction in the cluster); and an 8-byte int64 timestamp (starting from January 1, 1970) recorded when the latest decoded transaction log or checkpoint log was generated. As for the end-of-message character, binary messages end with character 'F'. Text and JSON messages end with 0 when sent in batches and have no end-of-message character in other cases. The message content is transmitted in big-endian mode. The specific format is illustrated in the following figure. (For compatibility reasons, the LSN naming convention is retained; its actual meaning depends on the specific scenario.)



- **parallel-decode-num**:

  (Effective only for streaming decoding) Specifies the number of decoder threads used in parallel decoding. This option remains ineffective when system functions are called—only the value range is verified in such cases.

  Value type: integer.

  Value range: 1–20. **1** indicates that decoding follows the original serial logic. Other values indicate that parallel decoding is enabled.

  Default value: **1**.

NOTICE

When **parallel-decode-num** is unset (defaulting to **1**) or explicitly set to **1**, the options described in the subsequent "Parallel Decoding" section are not configurable.

- **output-order**:

  (Effective only for streaming decoding) Specifies whether to sort decoding results by CSN. This option remains ineffective when system functions are called—only the value range is verified in such cases.

  Value type: integer.

  Value range:

  – **0**: Decoding results are sorted by transaction commit LSN. This sorting is available only when the **confirmed_csn** of the decoding replication slot is **0** (indicating an LSN-based replication slot). Otherwise, an error is reported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **1**: Decoding results are sorted by transaction CSN. This sorting is available only when the **confirmed_csn** of the decoding replication slot is not **0**. Otherwise, an error is reported.

  Default value: **0**.

  NOTICE

  - When **output-order** is unset (defaulting to **0**, that is, sorting results by commit LSN) or explicitly set to **0**, the options described in the subsequent "Distributed Strong-Consistency Decoding" section are not configurable.
  - In streaming decoding scenarios, when a DN receives a logical decoding connection from a CN, the **output-order** option is ineffective, and decoding results are sorted by CSN by default.

- **auto-advance**:

  (Effective only for streaming decoding) Specifies whether to automatically update the logical replication slot.

  Value type: Boolean.

  Value range:

  - **true**: The logical replication slot is updated to the current decoding position when all sent logs are confirmed and there are no further transactions to send.
  - **false**: The logical replication slot is updated only when the replication service calls the log confirmation API.

  Default value: **false**.

- **skip-generated-columns**:

  (Logical decoding control parameter) Specifies whether to skip generating decoding results for stored generated columns. This parameter remains ineffective on old tuples (those that have undergone an UPDATE or DELETE operation), meaning that decoding results will still be generated for their stored generated columns. Currently, stored generated columns are not supported in distributed mode, and this parameter is not configurable for decoding on CNs.

  Value type: Boolean.

  Value range:

  - **true**/**on**: No decoding results are generated for stored generated columns.
  - **false**/**off**: Decoding results are generated for stored generated columns.

  Default value: **false**/**off**.

  Virtual generated columns are not controlled by this parameter. DML decoding results will not be generated for virtual generated columns.

- **enable-ddl-decoding**:

  (Logical decoding control parameter) Specifies whether to enable logical decoding for DDL statements.

  Value type: Boolean.

  Value range:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **true**: Logical decoding is enabled for DDL statements.
  - **false**: Logical decoding is disabled for DDL statements.

  Default value: **false**.

- **enable-ddl-json-format**:

  (Logical decoding control parameter) Specifies the DDL reverse parsing process and output format.

  Value type: Boolean.

  Value range:

  - **true**: DDL reverse parsing results are output in JSON format.
  - **false**: DDL reverse parsing results follow the format specified by **decode-style**. When **decode-style** is set to **'j'**, results are output in text format.

  Default value: **false**.

- **timezone-is-utc**:

  (Logical decoding control parameter) Specifies the output format of time values with time zones (for example, the timestamptz type in ORA- or MySQL-compatible mode, or timestamp type in M-compatible mode). This parameter takes effect only during streaming decoding and is ignored during function decoding.

  Value type: Boolean.

  Value range:

  - **true**: Time values are decoded and output in time zone 0.
  - **false**: Time values are decoded and output in the current database's time zone.

  Default value: **false**.

- **decode-sequence**:

  (Logical decoding control parameter) Specifies whether to generate decoding results for logs that record sequence value changes.

  Value type: Boolean.

  Value range:

  - **true**: Not available currently.
  - **false**: No decoding results are generated for logs that record sequence value changes.

  Default value: **false**.

  NOTICE

  Currently, **decode-sequence** can only be set to **false**. Attempting to set it to **true** will result in an error and cause the system to exit upon decoding.

- **enable-decode-position**:

  (Logical decoding control parameter) Specifies whether to send decoding position logs to the client when the logs are read during logical decoding.

  Value type: Boolean.

  Value range:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **true**: Position logs are sent.
  - **false**: Position logs are not sent.

  Default value: **false**.

  > NOTE
  >
  > The following explains how decoding position logs are parsed in parallel decoding scenarios: In binary format, the log message starts with character 'L' (indicating a decoding position log). It then includes the following content: an 8-byte uint64 CSN generated when the decoding position log was sent; an 8-byte uint64 LSN (for LSN-based decoding, representing the WAL position at the time the decoding position log was sent) or CSN (for CSN-based decoding, generated when the decoding position log was sent); and a 4-byte uint32 timestamp (starting from January 1, 1970) recorded when the system function gs_add_logical_decoding_position_xlog was called. As for the end-of-message character, binary messages end with character 'F'. Text and JSON messages end with 0 when sent in batches and have no end-of-message character in other cases. The specific format is illustrated in **Figure 7-4**.

  Figure 7-4 Heartbeat protocol format

- **white-table-list1**, **white-table-list2**, **white-table-list3**, **white-table-list4**, **white-table-list5**, **white-table-list6**, **white-table-list7**, **white-table-list8**, and **white-table-list9**:

  (Extended whitelist parameters) Specify the names of the schemas and tables to be decoded, functionally equivalent to **white-table-list**. These parameters take effect only during streaming decoding and are not supported during SQL function decoding.

  Value range: a string of whitelisted table names, separated by commas (,). You can use an asterisk (*) for fuzzy matching of all. Separate schema names and table names with periods (.), and avoid including any whitespace.

- **data-limit**:

  Controls the volume of data output by logical decoding.

  When set in the GUC parameter **logical_decode_options_default**, the value must be an integer in the range [0,100], measured in GB. The default value is **10**. A value of **0** indicates no size limit for the decoding result.

  The GUC parameter must be used together with the input parameter **data-limit** of the pg_logical_get_area_changes function. For details, see the pg_logical_get_area_changes function in "SQL Reference > Functions and Operators > System Administration Functions > Logical Replication Functions" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **enable-rowno**:

  (Logical decoding control parameter) Specifies whether to output the **rowno** system column for new and old tuples in INSERT/UPDATE/DELETE operations when decoding a primary key–free table that includes the **rowid** system column and has the **REPLICA IDENTITY** option set to **FULL** or any other value equivalent to **FULL**. When using this option to output the **rowno** system column, ensure that the target database contains columns with the same name and type. Otherwise, logical decoding logs may fail to be replayed. If you have any questions, contact Huawei technical support.

  Value type: Boolean.

  Value range:

  – **true**: The **rowno** system column is output.

  – **false**: The **rowno** system column is not output.

  Default value: **false**.

  NOTICE

  - This option is ineffective for the sql_decoding plug-in.
  - "**REPLICA IDENTITY** equivalent to **FULL**" refers to cases where the **NOTHING** or **DEFAULT** option is set for a Ustore table. For details, see "SQL Reference > SQL Syntax > A > ALTER TABLE" in *Reference*.

- enable-cn-decode

  (Logical decoding control parameter) Specifies whether to enable CN decoding. CN decoding is disabled by default, necessitating decoding on directly connected DNs.

  Value type: Boolean.

  Value range:

  – **true**/**on**: CN decoding is enabled.

  – **false**/**off**: CN decoding is disabled.

  Default value: **false**/**off** (default for newly installed databases). This parameter is introduced in version 507.0.0. After an upgrade from a pre-507.0.0 version, this parameter defaults to **false**/**off**. If CN decoding tasks exist before the upgrade, enable this parameter after the upgrade. Otherwise, an error will occur when the CN decoding tasks are started.

  NOTICE

  This option can only be set in the GUC parameter **logical_decode_options_default** and is disabled by default. To enable it, contact Huawei technical support.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Distributed Strong-Consistency Decoding

NOTICE

In the current version, use decoding on directly connected DNs because the distributed strong-consistency decoding (CN decoding) feature is disabled by default. If required, contact Huawei technical support to enable the **enable-cn-decode** parameter after evaluation.

- **logical-receiver-num**:

  (Effective only for streaming decoding) Specifies the number of logical receiver threads started for distributed decoding. This option remains ineffective when system functions are called—only the value range is verified in such cases.

  Value type: integer.

  Value range: 1–20.

  Default value: **1**. If set higher than the number of shards in the current cluster, the value is automatically adjusted to match the shard count.

- **slice-id**:

  (Configurable only for decoding on connected DNs) Specifies the ID of the shard where the current DN resides. This option is used to decode replication tables.

  Value type: integer.

  Value range: 0–8192.

  Default value: **-1**, indicating that no shard ID is specified. In the default configuration, an error will be reported during replication table decoding.

CAUTION

This option can be used to decode on a connected DN by using a CSN-based logical replication slot (a slot with a non-zero **confirmed_csn**). It specifies the DN shard ID (SN, which starts from **0** for the first shard). If left unset (the default value **-1** is applied), an error will be reported during replication table decoding. This option also applies when a CN is connected for distributed decoding and collects decoding results from DNs. In this scenario, the CN collects replication table decoding results only from the specified DN shard. In this scenario, you are advised not to manually connect to DNs for decoding.

- **start-position**:

  (Configurable only for decoding on connected DNs) Filters out transactions with CSNs less than the specified CSN. For the transaction with the specified CSN, it also filters out logs with LSNs less than the specified LSN. Note that BEGIN logs of the transaction with the specified CSN are guaranteed to be filtered out.

  Value type: string.

  Value range: two uint64 values separated by a slash (/), where the left and right values represent the CSN and LSN, respectively.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **CAUTION**
>
> During CN-side decoding where a CN connects to a DN and sends a decoding request, you can configure this option to filter logs that may have already been received. In this scenario, you are advised not to use this option when manually connecting to DNs for decoding.

Serial Decoding

- **force-binary**:

  Specifies whether to output decoding results in binary format. The actual behavior depends on the specific scenario.

  - For the system functions pg_logical_slot_get_binary_changes and pg_logical_slot_peek_binary_changes:

    Value range: Boolean. Default value: **false**. This setting has no effect because decoding results are always output in binary format.
  - For system functions pg_logical_slot_get_changes, pg_logical_slot_peek_changes, and pg_logical_get_area_changes:

    Value range: Boolean (**false** only). Decoding results are output in text format.
  - For streaming decoding (supported only when DNs are connected):

    Value range: Boolean. Default value: **false**. This setting has no effect because decoding results are always output in text format.

Parallel Decoding

The following options are configurable only for streaming decoding:

- **decode-style**:

  When **enable-ddl-json-format** is set to **true**, the decoding format of DDL statements is controlled by **enable-ddl-json-format**, and **decode-style** specifies only the decoding format of DML statements. When **enable-ddl-json-format** is set to **false**, **decode-style** specifies the decoding format for both DML and DDL statements. When **decode-style** is set to **'j'**, DDL statements are output in text format, whereas DML statements follow the JSON format specified by **decode-style**.

  Value range: char-type values **'j'**, **'t'**, and **'b'**, representing the JSON, text, and binary formats, respectively.

  Default value:

  - When **decode-style** is not specified:

    For the replication slot plug-ins mppdb_decoding and sql_decoding, **decode-style** defaults to **'b'** (binary format). For the replication slot plug-ins parallel_binary_decoding, parallel_json_decoding, and parallel_text_decoding, **decode-style** defaults to **'b'** (binary format), **'j'** (JSON format), and **'t'** (text format) respectively.
  - When **decode-style** is specified:

    Decoding is performed based on the specified **decode-style**.

  For the JSON and text formats, in batch-sent decoding results: The uint32 comprising the first four bytes in each decoding statement indicates the total

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

byte count of the statement (excluding the four bytes occupied by the uint32 itself, with **0** signifying the end of the current decoding batch); and an 8-byte uint64 indicates the LSN (corresponding to **first_lsn** in the BEGIN scenario, **end_lsn** in the COMMIT scenario, and **lsn** in other scenarios).

Take the mppdb_decoding plug-in as an example. When **decode-style** is set to **'b'**, decoding is performed in binary format, producing results as follows:

```
current_lsn: 0/CFE5C80 BEGIN CSN: 2357 first_lsn: 0/CFE5C80
current_lsn: 0/CFE5D40 INSERT INTO public.test1 new_tuple: {a[typid = 23]: "1", b[typid = 23]: "2"}
current_lsn: 0/CFE5E68 COMMIT xid: 78108
```

When **decode-style** is set to **'j'**, decoding is performed in JSON format, producing results as follows:

```
BEGIN CSN: 2358 first_lsn: 0/CFE6220
{"table_name":"public.test1","op_type":"INSERT","columns_name":["a","b"],"columns_type":
["integer","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":[],"old_keys_val":[]}
COMMIT XID: 78109
```

When **decode-style** is set to **'t'**, decoding is performed in text format, producing results as follows:

```
BEGIN CSN: 2359 first_lsn: 0/CFE64D0
table public test1 INSERT: a[integer]:3 b[integer]:4
COMMIT XID: 78110
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

The binary encoding rules are as follows:

1. The first four bytes indicate the total byte count of the decoding result that follows—up to (but excluding) either the statement-level delimiter 'P' or the end-of-batch marker 'F'. A value of **0** signals the end of the current decoding batch.
2. The next 8-byte uint64 indicates the LSN (corresponding to **first_lsn** in the BEGIN scenario, **end_lsn** in the COMMIT scenario, and **lsn** in other scenarios).
3. The next 1-byte letter may be **B**, **C**, **I**, **U**, or **D**, representing BEGIN, COMMIT, INSERT, UPDATE, or DELETE, respectively.
4. If the letter in step **c** is **B**:
    1. The next 8-byte uint64 indicates the CSN.
    2. The subsequent 8-byte uint64 indicates **first_lsn**.
    3. (Optional) If the next 1-byte letter is **T**, the following 4-byte uint32 indicates the length of the transaction commit timestamp, and the subsequent characters of that length form the timestamp string.
    4. (Optional) If the next 1-byte letter is **N**, the following 4-byte uint32 indicates the length of the transaction username, and the subsequent characters of that length represent the username.
    5. Additional decoding statements may still follow, separated by a 1-byte letter **P** or **F**. **P** indicates that more statements remain to be decoded in the current batch, whereas **F** marks the end of the batch.
5. If the letter in step **c** is **C**:
    1. (Optional) If the next 1-byte letter is **X**, the following 8-byte uint64 indicates the XID.
    2. (Optional) If the next 1-byte letter is **T**, the following 4-byte uint32 indicates the timestamp length, and the subsequent characters of that length form the timestamp string.
    3. When logs are sent in batches, decoding a COMMIT log does not necessarily indicate the end of the batch. Additional transaction decoding results may still follow. If the next 1-byte letter is **P**, more transactions remain to be decoded in the current batch, whereas **F** marks the end of the batch.
6. If the letter in step **c** is **I**, **U**, or **D**:
    1. The next 2-byte uint16 indicates the length of the schema name.
    2. The system reads the schema name based on this length.
    3. The next 2-byte uint16 indicates the length of the table name.
    4. The system reads the table name based on this length.
    5. (Optional) If the next 1-byte letter is **N**, it denotes a new tuple. If the letter is **O**, it denotes an old tuple. New tuples are sent first.
        1. The next 2-byte uint16 indicates the number of columns to be decoded for the tuple, which is recorded as **attrnum**.
        2. The following procedure is repeated for *attrnum* times:
            1. The next 2-byte uint16 indicates the length of the column name.
            2. The system reads the column name based on this length.
            3. The next 4-byte uint32 indicates the OID of the current column type.
            4. The next 4-byte uint32 indicates the length of the current column value (stored in string format). A value of **0xFFFFFFFE** identifies a **rowno** system column (actual length: 8 bytes). **0xFFFFFFFF** denotes NULL, and **0** denotes a zero-length string.
            5. The system reads the column value based on this length. The **rowno** system column is read as 8 bytes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6. Additional decoding statements may still follow, separated by a 1-byte letter **P** or **F**. **P** indicates that more statements remain to be decoded in the current batch, whereas **F** marks the end of the batch.

- **sending-batch**:

  Specifies whether to send decoding results in batches.

  Value type: integer.

  Value range:

  – **0**: Decoding results are sent one by one.

  – **1**: Decoding results are sent in batches when they reach 1 MB.

  Default value: **0**.

  When **sending-batch** is enabled with the decoding format set to **'j'** or **'t'**, each original decoding statement will be preceded by a uint32 value indicating the length of the decoding result (excluding the bytes occupied by the uint32 itself) and a uint64 value indicating the LSN of the result.

NOTICE

In CSN-based decoding scenarios (where **output-order=1**), **sending-batch** applies only within individual transactions, allowing multiple small statements from the same transaction to be grouped into a batch. However, transactions are not batched together, nor are BEGIN/COMMIT statements.

- **parallel-queue-size**:

  Specifies the length of the queue used for interaction between parallel logical decoder threads.

  Value type: integer.

  Value range: integral powers of 2 between 2 and 1024.

  Default value: **128**.

  The queue length is positively proportional to memory usage during decoding.

- **max-logical-decode-memory**:

  (Memory control parameter for parallel decoding) Specifies the total memory (including cache) that can be allocated to a parallel decoding task. If this limit is exceeded, the currently decoded transaction will be flushed to disk. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding.

  Value type: integer.

  Unit: GB.

  Value range: 6–1024.

  Default value: **6**.

  When a decoding task starts, the upper limit is controlled by the **max-control-memory-size** parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- In JSON decoding format, memory usage may temporarily increase due to the tree structure of JSON. Therefore, it is advisable to set this limit to 10 GB. If set to 6 GB, brief memory overshoots are expected. After decoding is complete, temporary memory will be released, and the total memory usage will not exceed **max-logical-decode-memory**. For details about the decoding format, see **decode-style**.
- When processing TOAST data, it is advised to set the logical decoding memory parameter as follows: **max-logical-decode-memory** = Thread concurrency × 1 GB + 10 GB. If **max-logical-decode-memory** exceeds **max-control-memory-size**, adjust **max-control-memory-size** in the GUC parameter **logical_decode_options_default**.
- This limit should ideally be ≤ 50% of **max_dynamic_memory** to prevent excessive memory allocation that could hinder other online services.

- **max-control-memory-size**:

  Specifies the upper limit for **max-logical-decode-memory**, a memory control parameter for parallel decoding. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding.

  Value type: integer.

  Unit: GB.

  Value range: 6–1024.

  Default value: **10**.

  If the startup parameter **max-logical-decode-memory** exceeds this limit, it will be automatically adjusted when a decoding task starts, with a corresponding message recorded in system logs.

NOTICE

- The decoding option **max-control-memory-size** can only be set within the GUC parameter **logical_decode_options_default**. Attempting to set it during the startup of a decoding task will result in an error and cause the system to exit.
- This limit should ideally be ≤ 50% of **max_process_memory** to prevent memory overflow during decoding tasks.

- **max-toast-memory-size**:

  Specifies the memory limit for parallel decoding of TOAST data, specifically the maximum memory allocated for reading TOAST fields in Xlogs. If this limit is exceeded, the TOAST data of the currently decoded transaction will be flushed to disk. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding.

  Value type: integer.

  Unit: MB.

  Value range: 100–1048576.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Default value: **100**.

NOTICE

The decoding option **max-toast-memory-size** can only be set within the GUC parameter **logical_decode_options_default**. Attempting to set it during the startup of a decoding task will result in an error and cause the system to exit.

- **max-queue-cache-num**:

  Specifies the upper limit for the parallel decoding parameter **max-decode-to-sender-cache-num**. This parameter takes effect only for parallel decoding on connected DNs and remains ineffective for distributed strong-consistency decoding, serial decoding, and SQL function decoding.

  Value type: integer.

  Unit: N/A.

  Value range: 8192–65535.

  Default value: **16384**.

  If the startup parameter **max-decode-to-sender-cache-num** exceeds this limit, it will be automatically adjusted when a decoding task starts, with a corresponding message recorded in system logs.

NOTICE

The decoding option **max-queue-cache-num** can only be set within the GUC parameter **logical_decode_options_default**. Attempting to set it during the startup of a decoding task will result in an error and cause the system to exit.

7.7.1.3 Logical Decoding by SQL Functions

GaussDB can call SQL functions to create, delete, and update logical replication slots, and obtain decoded transaction logs.

Procedure

Step 1 Log in to any primary DN in the GaussDB cluster as a user with the REPLICATION permission.

Step 2 Run the following command to connect to the database through a DN port:

```
gsql -U user1 -W password -d gaussdb -p 40000 -r
```

In the preceding command, *user1* represents the username, *password* represents the user's password, *gaussdb* represents the name of the database to connect to, and *40000* represents the DN port number. Replace them with the actual values. Replication slots are created on DNs. Therefore, you need to connect to the database through a DN port.

Step 3 Create a logical replication slot named **slot1**.

```
gaussdb=> SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
slotname | xlog_position
----------+---------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
slot1    | 0/601C150
(1 row)
```

Step 4 Connect to the database through a CN port, create a table named **t** in the database, and insert data into **t**.

```
gaussdb=> CREATE TABLE t(a int PRIMARY KEY, b int);
gaussdb=> INSERT INTO t VALUES(3,3);
```

Step 5 Connect to the DN (refer to **Step 2**) and read the decoding result of **slot1**. The number of decoded entries is 4,096.

NOTE

For details about the logical decoding options, see **Logical Decoding Options**.

```
gaussdb=> SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096);
location  |  xid  | data
-----------+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------
-------------------------------------------
 0/601C188 | 1010023 | BEGIN 1010023
 0/601ED60 | 1010023 | COMMIT 1010023 (at 2023-09-14 16:03:51.394287+08) CSN 1010022
 0/601ED60 | 1010024 | BEGIN 1010024
 0/601ED60 | 1010024 | {"table_name":"public.t","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
 0/601EED8 | 1010024 | COMMIT 1010024 (at 2023-09-14 16:03:57.239821+08) CSN 1010023
(5 rows)
```

Step 6 Drop the logical replication slot **slot1**.

```
gaussdb=> SELECT * FROM pg_drop_replication_slot('slot1');
 pg_drop_replication_slot
--------------------------

(1 row)
```

----End

7.7.1.4 Logical Data Replication Using Streaming Decoding

Through streaming logical decoding, third-party replication tools can extract logical logs from GaussDB and replay them on a peer database.

For the implementation code of replication tools that use JDBC to connect to databases, refer to **Logical Replication (Non-Multi-Primary)**.

7.7.1.5 Logical Decoding of DDL Statements

DDL statements can be properly executed on GaussDB and obtained using a logical decoding tool.

Table 7-5 Supported DDL statements

Table	CREATE TABLE [PARTITION | AS | SUBPARTITION], ALTER TABLE [PARTITION | SUBPARTITION], DROP TABLE, RENAME TABLE, and TRUNCATE
Index	CREATE INDEX, ALTER INDEX, DROP INDEX, and REINDEX
User-defined Function	CREATE FUNCTION, ALTER FUNCTION, and DROP FUNCTION

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

User-defined Stored Procedure	CREATE PROCEDURE, ALTER PROCEDURE, and DROP PROCEDURE
Trigger	CREATE TRIGGER, ALTER TRIGGER, and DROP TRIGGER
Sequence	CREATE SEQUENCE, ALTER SEQUENCE, and DROP SEQUENCE
Schema	CREATE SCHEMA, ALTER SCHEMA, and DROP SCHEMA
Comment On	COMMENT ON

Table 7-6 DDL statements supported by M-compatible databases

Table	ALTER TABLE, CREATE TABLE, DROP TABLE, ALTER TABLE PARTITION, CREATE TABLE PARTITION, and TRUNCATE
Index	ALTER INDEX, CREATE INDEX, DROP INDEX, REINDEX, ALTER TABLE DROP INDEX, and ALTER TABLE ADD INDEX
Sequence	ALTER SEQUENCE, CREATE SEQUENCE, and DROP SEQUENCE
Schema	ALTER SCHEMA, CREATE SCHEMA, DROP SCHEMA, ALTER DATABASE, CREATE DATABASE, and DROP DATABASE
Comment On	COMMENT ON

Description

When DML operations are performed in a database, the storage engine generates corresponding DML logs for potential restoration purposes. Specifically, by decoding these DML logs, the engine can reconstruct the original DML statements and generate logical logs. In contrast, for DDL statements, the database does not log the original DDL statements. Instead, it records DML logs that reflect changes made to system catalogs as a result of the DDL statements. Given the wide variety and syntactic complexity of DDL statements, reconstructing the original DDL statements from the DML logs of system catalogs is highly challenging for logical replication. To overcome this limitation, the database now additionally records DDL logs to preserve the original DDL information. This allows the original DDL statements to be reconstructed directly from the DDL logs during decoding.

During DDL statement execution, the SQL engine's parser analyzes the syntax and lexicon of the original statement to generate a parse tree. (Different DDL syntaxes yield different types of parse trees, each containing complete DDL statement information.) Based on this information, the executor carries out the necessary operations to generate or modify metadata.

The following explains how the introduction of DDL logs facilitates logical decoding of DDL statements. DDL log content is derived from both the parser result (parse tree) and the executor result, with the logs being generated after the executor completes its operations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DDL statements are reverse-parsed from their parse trees. This reverse parsing process converts DDL commands into JSON-formatted statements and provides the information needed to reconstruct DDL commands at the target location. Compared to the original DDL command strings, DDL reverse parsing offers the following benefits:

1. Each parsed database object is associated with a schema, eliminating ambiguity even when different search paths are used.
2. The structured JSON statements and formatted output are compatible with heterogeneous databases. If different database versions with varying DDL syntaxes are involved, these differences must be resolved beforehand.

The reverse-parsing result is output in a normalized format. While equivalent to the original user input, it may not be identical. For example:

Example 1: If a function body contains no single quotation marks ('), any delimiters ($$) in the function body will be parsed as single quotation marks (').

Original SQL statements:

```
CREATE FUNCTION func(a INT) RETURNS INT AS
$$
BEGIN
a:= a+1;
CREATE TABLE test(col1 INT);
INSERT INTO test VALUES(1);
DROP TABLE test;
RETURN a;
END;
$$
LANGUAGE plpgsql;
```

Reverse parsing result:

```
CREATE  FUNCTION public.func ( IN a pg_catalog.int4 ) RETURNS pg_catalog.int4 LANGUAGE plpgsql
VOLATILE CALLED ON NULL INPUT SECURITY INVOKER COST 100 AS '
BEGIN
a:= a+1;
CREATE TABLE test(col1 INT);
INSERT INTO test VALUES(1);
DROP TABLE test;
RETURN a;
END;
';
```

Example 3: **ALTER INDEX "Alter_Index_Index" REBUILD PARTITION "CA_ADDRESS_SK_index2"** is reverse-parsed as **REINDEX INDEX public."Alter_Index_Index" PARTITION "CA_ADDRESS_SK_index2"**.

Example 4: When you create or modify a range partitioned table, the START END syntax is decoded and converted into the LESS THAN syntax.

```
gaussdb=# CREATE TABLE test_create_table_partition2 (c1 INT, c2 INT)
PARTITION BY RANGE (c2) (
   PARTITION p1 START(1) END(1000) EVERY(200) ,
   PARTITION p2 END(2000),
   PARTITION p3 START(2000) END(2500),
   PARTITION p4 START(2500),
   PARTITION p5 START(3000) END(5000) EVERY(1000)
);
```

Reverse parsing result:

```
gaussdb=# CREATE TABLE test_create_table_partition2 (c1 INT, c2 INT)
PARTITION BY RANGE (c2) (
   PARTITION p1_0 VALUES LESS THAN ('1'), PARTITION p1_1 VALUES LESS THAN ('201'), PARTITION p1_2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
VALUES LESS THAN ('401'), PARTITION p1_3 VALUES LESS THAN ('601'), PARTITION p1_4 VALUES LESS
THAN ('801'), PARTITION p1_5 VALUES LESS    THAN ('1000'),
    PARTITION p2 VALUES LESS THAN ('2000'),
    PARTITION p3 VALUES LESS THAN ('2500'),
    PARTITION p4 VALUES LESS THAN ('3000'),
    PARTITION p5_1 VALUES LESS THAN ('4000'),
    PARTITION p5_2 VALUES LESS THAN ('5000')
);
```

Example 5: When adding a column to a table, you can use IF NOT EXISTS for judgment.

- Original SQL statements:

  ```
  gaussdb=# CREATE TABLE IF NOT EXISTS tb5 (c1 int,c2 int) with (ORIENTATION=ROW,
  STORAGE_TYPE=USTORE);
  gaussdb=# ALTER TABLE IF EXISTS tb5 * ADD COLUMN IF NOT EXISTS c2 char(5) after c1; --
  Decodable. This statement skips the int-type c2 column in the table, and the reverse parsing result
  shows the original type for the c2 column.
  ```

  Reverse parsing result:

  ```
  gaussdb=# ALTER TABLE IF EXISTS public.tb5 ADD COLUMN IF NOT EXISTS c2 pg_catalog.int4 AFTER
  c1;
  ```

- Original SQL statement:

  ```
  gaussdb=# ALTER TABLE IF EXISTS tb5 * ADD COLUMN IF NOT EXISTS c2 char(5) after c1,  ADD
  COLUMN IF NOT EXISTS c3 char(5) after c1; -- Decoded. The type of the new column c3 in the
  reverse parsing result is correct.
  ```

  Reverse parsing result:

  ```
  gaussdb=# ALTER TABLE IF EXISTS public.tb5 ADD COLUMN IF NOT EXISTS c2 pg_catalog.int4 AFTER
  c1, ADD COLUMN IF NOT EXISTS c3 pg_catalog.bpchar(5) AFTER c1;
  ```

- Original SQL statement:

  ```
  gaussdb=# ALTER TABLE IF EXISTS tb5 * ADD COLUMN c2 char(5) after c1,  ADD COLUMN IF NOT
  EXISTS c4 int after c1; -- Not decoded. An error occurs when the statement is executed.
  ```

Example 6: When a database object is defined (DROP TABLE/INDEX/SEQUENCE), the **IFEXISTS** option is added to the decoding result.

- Original SQL statements:

  ```
  gaussdb=# CREATE TABLE IF NOT EXISTS tb6 (c1 int,c2 int) with (ORIENTATION=ROW,
  STORAGE_TYPE=USTORE);
  gaussdb=#   DROP TABLE tb6;
  ```

  Reverse parsing result:

  ```
  gaussdb=# DROP TABLE IF EXISTS public.tb6 RESTRICT;
  ```

Specifications and Limitations

- Logical decoding specifications for DDL statements:
  - In a standard environment, logical decoding performance reaches approximately 100 MB/s in DDL-only scenarios, and also about 100 MB/s for DDL/DML hybrid transactions.
  - Enabling this feature (by setting **wal_level=logical** and **enable_logical_replication_ddl=on**) degrades the execution performance of DDL operations by no more than 15%.
- General limitations on decoding (serial or parallel):
  - DDL operations on local temporary objects cannot be decoded.
  - DDL statements on foreign tables cannot be decoded.
  - The **default** value in ALTER TABLE ADD COLUMN does not support stable or volatile functions. Similarly, CHECK expressions for columns in CREATE TABLE and ALTER TABLE are not applicable to stable or volatile functions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If an ALTER TABLE statement includes multiple sub-statements and any of them violates the aforementioned restrictions, the entire ALTER TABLE statement will not be decoded. (The **provolatile** column in the pg_proc system catalog helps you determine whether the **default** value and CHECK expressions are used for stable or volatile functions.)

```
-- Case 1:
gaussdb=# ALTER TABLE tbl_28 ADD COLUMN b1 TIMESTAMP DEFAULT NOW(); -- 's' is not decoded.
gaussdb=# ALTER TABLE tbl_28 ADD COLUMN b2 INT DEFAULT RANDOM(); -- 'v' is not decoded.
gaussdb=# ALTER TABLE tbl_28 ADD COLUMN b3 INT DEFAULT ABS(1); -- 'i' is decoded.

-- Case 2:
gaussdb=# CREATE DATABASE m_db with DBCOMPATIBILITY = 'M' template=templatem encoding 'UTF8';
m_db=# CREATE TABLE t1(id int) CHAR SET = 'GBK';
m_db=# ALTER TABLE t1 ADD COLUMN col1 varchar(100) DEFAULT '1'; -- In M-compatible mode, the pg_convert function is internally called to convert the character set. Since pg_convert is a stable function, the ALTER TABLE statement is not decoded.
```

- The distributed mode does not support DDL decoding for CREATE MATERIALIZED VIEW.
- The system does not support DDL decoding for CREATE/ALTER/DROP VIEW or COMMENT ON VIEW.
- The system does not support DDL decoding for REINDEX DATABASE/SYSTEM.
- The system does not support trigger-related DDL decoding on views.
- The system does not support DDL decoding for CONCURRENTLY statements.
- When IF NOT EXISTS is present in the statement for creating an object but the object already exists, the statement is not decoded. When IF EXISTS is present in the statement for dropping an object but the object does not exist, the statement is not decoded.
- The ALTER PACKAGE COMPILE statement itself is not decoded; however, DDL/DML statements within the instantiated package content are decoded. If the instantiated package content contains no DDL or DML statements, ALTER PACKAGE COMPILE will be ignored during logical decoding.
- Only basic DDL syntax is supported.
- Logical decoding does not support hybrid transactions that mix DDL (or DCL) with DML statements. In such hybrid transactions, DML statements that follow DDL statements cannot be decoded. A transaction is strictly classified as hybrid if either the main transaction or any subtransaction includes DDL operations—in such cases, subsequent DML operations will not be decoded.

```
-- No reverse parsing is performed for DCL statements, as they are not supported. DML statements that follow DCL statements are also not reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# GRANT ALL PRIVILEGES to u01;
gaussdb=# INSERT INTO test1(col1) values(1);
gaussdb=# COMMIT;

-- Only the first and third SQL statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# CREATE TABLE mix_tran_t4(id int);
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# CREATE TABLE mix_tran_t5(id int);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# COMMIT;

-- Only the first and second SQL statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# CREATE TABLE mix_tran_t6(id int);
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# COMMIT;

-- All statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# CREATE TABLE mix_tran_t7(id int);
gaussdb=# CREATE TABLE mix_tran_t8(id int);
gaussdb=# COMMIT;

-- Only the first and third SQL statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# CREATE TABLE mix_tran_t7(id int);
gaussdb=# CREATE TYPE compfoo AS (f1 int, f2 text);
gaussdb=# CREATE TABLE mix_tran_t8(id int);
gaussdb=# COMMIT;

-- All statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# COMMIT;

-- Only the first SQL statement is reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# CREATE TYPE compfoo AS (f1 int, f2 text);
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# COMMIT;

-- Only the first and third SQL statements are reverse-parsed.
gaussdb=# BEGIN;
gaussdb=# INSERT INTO mix_tran_t4 VALUES(111);
gaussdb=# CREATE TYPE compfoo AS (f1 int, f2 text);
gaussdb=# CREATE TABLE mix_tran_t9(id int);
gaussdb=# COMMIT;

-- If sub-transactions in a main transaction include DDL operations and are rolled back, no
operations will be parsed.
gaussdb=# BEGIN;
gaussdb=# savepoint s1;
gaussdb=# CREATE TABLE t2(col int);
gaussdb=# INSERT INTO t2 VALUES(1);
gaussdb=# ROLLBACK TO s1;
gaussdb=# insert INTO t1 VALUES(2);
gaussdb=# COMMIT;
```

- For the CREATE TABLE AS SELECT, SELECT INTO, and CREATE TABLE AS statements, only CREATE TABLE can be decoded; INSERT cannot be decoded.

  NOTE

  For tables created using CREATE TABLE AS SELECT, its ALTER and DROP statements remain decodable.

  Example:

  Original SQL statements:

```
CREATE TABLE IF NOT EXISTS tb35_2 (c1 int) with (storage_type=USTORE,ORIENTATION=ROW);
INSERT INTO tb35_2 VALUES (6);
CREATE TABLE tb35_1 with (storage_type=USTORE,ORIENTATION=ROW) AS SELECT * FROM
tb35_2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Reverse parsing result for the last SQL statement:

```
 CREATE  TABLE  public.tb35_1 (c1 pg_catalog.int4) WITH (storage_type = 'ustore', orientation = 'row', compression = 'no') NOCOMPRESS;
```

- When the stored procedure, function, or advanced package being executed includes—or combines with other statements in the same transaction to form—a DDL/DML hybrid transaction, the database decodes it according to the hybrid transaction principle.
- For SQL statements using slashes (/) as the terminator for stored procedures, functions, package specifications, and package bodies, the decoding results do not include the "/" character. When using the decoding results to replay data, third-party tools should handle it according to the specific situation.

  - Original SQL statement:

    ```
    CREATE OR REPLACE PROCEDURE p62_1(var62_1 int)
       IS
       BEGIN
          var62_1 := 5;
       END;
    /
    ```

    Reverse parsing result:

    ```
    CREATE OR REPLACE PROCEDURE public.p62_1 ( IN var62_1 pg_catalog.int4 ) VOLATILE CALLED ON NULL INPUT SECURITY INVOKER COST 100 AS  DECLARE  BEGIN
          var62_1 := 5;
    END ;
    ```

  - Original SQL statement:

    ```
    CREATE OR REPLACE FUNCTION proc_add2 (num1 in int, num2 in int, num3 out int)
    return int
    AS
    BEGIN
    num3 := num1 + num2;
    END;
    /
    ```

    Reverse parsing result:

    ```
    CREATE OR REPLACE FUNCTION public.proc_add2 ( IN num1 pg_catalog.int4, IN num2 pg_catalog.int4, OUT num3 pg_catalog.int4 ) RETURN pg_catalog.int4 VOLATILE CALLED ON NULL INPUT SECURITY INVOKER COST 100 AS  DECLARE
    BEGIN
    num3 := num1 + num2;
    END ;
    ```

  - Original SQL statement:

    ```
    CREATE TABLE tbl_test(uid varchar(6) PRIMARY KEY, area_id varchar(5), level int);
    -- Create a package specification.
    CREATE OR REPLACE PACKAGE pkg_test AS
       -- Public stored procedures can be called by external systems.
       PROCEDURE proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
       PROCEDURE proc_del_tbl_test(v_uid in varchar);
       PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
    END pkg_test;
    /
    -- Create a package body.
    CREATE OR REPLACE PACKAGE BODY pkg_test AS
       PROCEDURE proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
       BEGIN
          INSERT INTO tbl_test VALUES (v_uid, v_area_id, v_level);
       END;
       PROCEDURE proc_del_tbl_test(v_uid in varchar) AS
       BEGIN
          DELETE FROM tbl_test WHERE uid = v_uid;
       END;
    ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
    BEGIN
        UPDATE tbl_test SET area_id = v_area_id, level = v_level WHERE uid = v_uid;
    END;
    var4 int := 10;
    -- Instantiation starts.
BEGIN
    var4 := 10;
    dbe_output.print_line(var4);
END pkg_test;
/
```

Reverse parsing result:

```
CREATE  TABLE  public.tbl_test (uid pg_catalog."varchar"(6) NOT NULL, area_id
pg_catalog."varchar"(5), level pg_catalog.int4, CONSTRAINT tbl_test_pkey PRIMARY KEY
(uid)) NOCOMPRESS
CREATE OR REPLACE PACKAGE public.pkg_test AUTHID CURRENT_USER IS  -- The public
stored procedure can be called by external systems.
    PROCEDURE proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
    PROCEDURE proc_del_tbl_test(v_uid in varchar);
    PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
END  pkg_test;
CREATE OR REPLACE PACKAGE BODY public.pkg_test IS  PROCEDURE
proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
    BEGIN
        INSERT INTO tbl_test VALUES (v_uid, v_area_id, v_level);
    END;
    PROCEDURE proc_del_tbl_test(v_uid in varchar) AS
    BEGIN
        DELETE FROM tbl_test WHERE uid = v_uid;
    END;
    PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
    BEGIN
        UPDATE tbl_test SET area_id = v_area_id, level = v_level WHERE uid = v_uid;
    END;
    var4 int := 10;
    -- Instantiation starts.
BEGIN
    var4 := 10;
    dbe_output.print_line(var4);
END
 pkg_test;
```

  - Logical decoding does not support the ledger database feature because the decoding result of the DDL statement for creating a ledger database contains hash columns.

    - Original statements:

      ```
      CREATE SCHEMA blockchain_schema WITH BLOCKCHAIN;
      CREATE TABLE blockchain_schema.blockchain_table(mes int);
      ```

    - Decoding result:

      ```
      CREATE SCHEMA blockchain_schema WITH BLOCKCHAIN;
      CREATE TABLE blockchain_schema.blockchain_table (mes pg_catalog.int4, hash_a1d895
      pg_catalog.hash16);  -- This statement cannot be directly replayed on the target end.
      ```

      To replay the statement, first manually disable the tamper-proof attribute of blockchain_schema on the target end, which converts blockchain_table into an ordinary table. Then re-execute the DML command.

      SQL commands:

      ```
      ALTER SCHEMA blockchain_schema WITHOUT BLOCKCHAIN;
      CREATE TABLE blockchain_schema.blockchain_table (mes pg_catalog.int4, hash_a1d895
      pg_catalog.hash16);
      ```

- In MySQL-compatible mode, the ALTER SCHEMA schema_name WITHOUT/WITH BLOCKCHAIN syntax cannot be reverse-parsed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Serial logical decoding limitations specific to DDL:
  - The sql_decoding plug-in does not support DDL statements in JSON format.
- Logical decoding does not apply to schemas created by CREATE USER.

Decoding Formats

- JSON format

  When a DDL statement is input, the SQL engine's parser analyzes its syntax and lexicon to generate a parse tree. Nodes in this tree contain complete DDL statement information. Based on the parse tree, the executor modifies metadata as required. After the executor completes its operations, the search path of the DDL object is provided. Upon successful execution of the executor, this feature performs reverse parsing of both the parse tree information and the executor result to reconstruct all information about the original DDL statement. Through reverse parsing, the entire DDL statement can be decomposed and output in JSON format to adapt to heterogeneous databases.

  After lexical and syntax analysis of the CREATE TABLE statement, a CreateStmt parse tree node is generated, containing table information, column information, distribution information (DistributeBy structure), and partition information (PartitionState structure). Once reverse parsing is completed, the result is output in JSON format as follows:

  ```
  {"JDDL":{"fmt":"CREATE %{persistence}s TABLE %{if_not_exists}s %{identity}D %{table_elements}s %{with_clause}s %{compression}s","identity":{"object_name":"test_create_table_a","schema_name":"public"},"compression":"NOCOMPRESS","persistence":"","with_clause":{"fmt":"WITH (%{with:, }s)","with":[{"fmt":"%{label}s = %{value}L","label":{"fmt":"%{label}I","label":"orientation"},"value":"row"},{"fmt":"%{label}s = %{value}L","label":{"fmt":"%{label}I","label":"compression"},"value":"no"}]},"if_not_exists":"","table_elements":{"fmt":"(%{elements:, }s)","elements":[{"fmt":"%{name}I %{column_type}T","name":"a","column_type":{"typmod":"","typarray":false,"type_name":"int4","schema_name":"pg_catalog"}}]}}}
  ```

  As shown above, the JSON output contains the search path of the object, where **identity** indicates that the schema is **public** and the table name is **test_create_table_a**. *%{persistence}s* corresponds to the following field (which remains empty in this example, as the current SQL statement does not include it):

  ```
  [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ]
  ```

  %{if_not_exists}s corresponds to the following field (which remains empty in this example, as the current SQL statement does not include it):

  ```
  [ IF NOT EXISTS ]
  ```

  %{identity}D corresponds to the following field in the SQL statement:

  ```
  table_name
  ```

  %{table_elements}s corresponds to the following field in the SQL statement:

  ```
  (column_name data_type)
  ```

  %{with_clause}s corresponds to the following field in the SQL statement:

  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  ```

  %{compression}s corresponds to the following field in the SQL statement:

  ```
  [ COMPRESS | NOCOMPRESS ]
  ```

- Format specified by **decode-style**

  The output format is controlled by the **decode-style** parameter. For example, when **decode-style** is set to **'j'**, the output format is as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
{"TDDL":"CREATE  TABLE  public.test_create_table_a (a pg_catalog.int4) WITH (orientation = 'row', compression = 'no') NOCOMPRESS"}
```

The statement also contains the schema name.

API Design

- New control parameters
  a. New logical decoding control parameters are introduced to manage the DDL reverse parsing process and output format. To enable them, you can use JDBC APIs or pg_logical_slot_peek_changes.
     - **enable-ddl-decoding**: The default value is **false**, indicating that logical decoding of DDL statements is disabled. To enable the decoding, set this parameter to **true**.
     - **enable-ddl-json-format**: The default value is **false**, indicating that the reverse parsing results of DDL statements are output in text format. To output the results in JSON format, set this parameter to **true**.
  b. A GUC parameter is introduced.
     - **enable_logical_replication_ddl**: The default value is **ON**, indicating that logical replication is supported for DDL statements. If set to any other value, DDL support is disabled. DDL execution results are reverse-parsed and WALs are generated only when this parameter is set to **ON**. Otherwise, no reverse parsing or WAL generation takes place.

       Logs related to **enable_logical_replication_ddl** can help determine whether the absence of DDL support in logical decoding results from user modification of this parameter.
- New logs

  The DDL log **xl_logical_ddl_message** (type: RM_LOGICALDDLMSG_ID) is added, which is defined as follows.

Name	Type	Description
db_id	OID	Database ID.
rel_id	OID	Table ID.
csn	CommitSeqNo	CSN-based snapshot.
cid	CommandId	Command ID.
tag_type	NodeTag	DDL type.
message_size	Size	Length of the log content.
filter_message_size	Size	Length of log information filtered by whitelist.
message	char *	DDL content.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Procedure

Step 1 To use the logical decoding feature, set the GUC parameter **wal_level** to **logical**. This parameter takes effect only after a cluster restart. For details about the **wal_level** parameter, see "Database GUC Parameters > GUC Parameters > Write Ahead Log > Settings" in *Reference*.

Step 2 Log in to the primary node of GaussDB as a user with the REPLICATION permission and run the following command to connect to the database:

```
gsql -U user1 -W password -d db1 -p 16000 -r
```

In the preceding command, *user1* represents the username, *password* represents the user's password, *db1* represents the name of the database to connect to, and *16000* represents the database port number. Replace them with the actual values.

Step 3 Create a logical replication slot named **slot1**.

```
gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
 slotname | xlog_position
----------+---------------
 slot1    | 0/3764C788
(1 row)
```

Step 4 Create a package in the database.

```
gaussdb=# CREATE OR REPLACE PACKAGE ldp_pkg1 IS
    var1 int:=1;  -- Public variable.
    var2 int:=2;
    PROCEDURE testpro1(var3 int);  -- Public stored procedure, which can be called by external systems.
END ldp_pkg1;
/
```

Step 5 Read the decoding result of **slot1**. You can use JDBC APIs or pg_logical_slot_peek_changes to update the replication slot.

NOTE

- For details about the logical decoding options, see **Logical Decoding Options** and new control parameters.
- During parallel decoding, you can modify the **decode_style** parameter through JDBC APIs to adjust the decoding format.

  The configuration option **decode-style** specifies the decoding format. It can be set to the char-type value **'j'**, **'t'** or **'b'**, representing the JSON, text, or binary format, respectively.

```
gaussdb=# SELECT data FROM pg_logical_slot_peek_changes('slot1', NULL, NULL, 'enable-ddl-decoding', 'true', 'enable-ddl-json-format', 'false') WHERE data not like 'BEGIN%' AND data not like 'COMMIT%' AND data not like '%dbe_pldeveloper.gs_source%';

                                                          data

-------------------------------------------------------------------------------------------------------------------------
------------------------------------------ {"TDDL":"CREATE OR REPLACE PACKAGE public.ldp_pkg1 AUTHID CURRENT_USER IS var1 int:=1; -- Public variable\n    var2 int:=2;\n
PROCEDURE testpro1(var3 int);  -- Public stored procedure, which can be called externally.\nEND  ldp_pkg1;\n /"}
(1 row)
```

Step 6 Drop the logical replication slot **slot1** and package **ldp_pkg1**.

```
gaussdb=# SELECT * FROM pg_drop_replication_slot('slot1');
 pg_drop_replication_slot
--------------------------

(1 row)

gaussdb=# DROP PACKAGE ldp_pkg1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  drop cascades to function public.testpro1(integer)
DROP PACKAGE
```

----End

7.7.1.6 Data Retrieval by Logical Decoding

Logical decoding can retrieve DML data on a single node. Specifically, it retrieves online or archived WALs.

- Online data: To retrieve online DML data, you can use the pg_logical_get_area_changes function. For details, see the usage of the pg_logical_get_area_changes function.
- Archived data: OM_Agent uses the pg_logical_get_area_changes function to retrieve archive logs on OBS. OM_Agent downloads archive logs from OBS to the corresponding node and creates a folder named with the first eight digits of the WAL file name. After the decoding is complete, the retrieved data is stored in a file and the file path is returned.

Limitations

1. The log level of parsed WALs is **logical**.
2. The **FULL** flag must be specified for copying a data table; otherwise, UPDATE and DELETE will modify rows without including all columns.
3. Do not perform VACUUM FULL—from the retrieval start point to the present—on service tables that have data changes recorded in WALs. Otherwise, you will be unable to retrieve data for the tables' DML operations that occurred before the VACUUM FULL.
4. The size of each WAL file cannot exceed 500 MB.
5. Xlogs generated before scale-out cannot be retrieved.
6. A result file is generated for each shard in the cluster. These files are not merged.
7. Only archived data can be retrieved. To retrieve the data, ensure that the archive function is enabled. If data has not been archived, it cannot be retrieved using this API.
8. Before downloading files, the OM Agent checks whether local space usage exceeds 80% of total capacity. If it does, the OM Agent reports "no enough space left on device, available space must be greater than 20%", indicating that additional space is required to store decoded files.
9. If the download or decoding fails, the downloaded WAL files are cleared. If the clearing fails, the program is not forcibly ended and only the error information is recorded in DN logs.
10. Users must specify a time within the [Minimum time,Maximum time] range supported by the gs_txn_lsn_time system catalog. An error will be reported if the specified start time exceeds the maximum supported time or if the end time is earlier than the minimum supported time.
11. If a node has been replaced, its pre-replacement data cannot be retrieved.
12. The data retrieval API cannot be concurrently called on the same node.
13. Data can only be retrieved for directly connected DNs.
14. If you have upgraded from an earlier version, you need to execute the SELECT gs_logical_dictionary_disabled() and SELECT gs_logical_dictionary_baseline()

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

functions to baseline data before using the data retrieval feature. You can retrieve only the Xlogs generated after this baseline; Xlogs generated before the upgrade cannot be parsed.

15. By default, the database can retrieve data generated within the past year and automatically deletes data older than that. To retrieve data older than one year, update the GUC parameter **logical_replication_dictionary_retention_time** before the data is cleared.
16. Data retrieval depends on the metadata of tables. If the metadata is abnormal, data retrieval may throw an error and exit.
17. Data retrieval cannot work at the transaction level; it supports only row-level retrieval. XIDs are printed when transactions are rolled back, requiring manual intervention. To retrieve data for DML statements in a hybrid transaction, you need to manually verify correctness.

7.8 Partitioned Table

This section systematically explains the semantics, principles, limitations, and other key aspects of partitioned tables to help you understand their query optimization and O&M capabilities in large-scale data scenarios.

7.8.1 Introduction to Table Partitioning

7.8.1.1 Background

As data volumes grow and usage scenarios diversify, databases increasingly handle larger capacities and more varied data types. Over the past 20 years, data sizes have expanded from megabytes and gigabytes to terabytes. To manage this scale and diversity, database management systems (DBMSs) must meet higher demands for data queries and data management. Consequently, databases must support various optimized search policies and O&M methods.

In classic computer science algorithms, people often use the Divide and Conquer method to solve large-scale problems. This approach breaks a complex problem into two or more identical or similar subproblems, which are further divided until they become simple enough to solve directly. The solution to the original problem can be regarded as the combination of the solutions to all smaller subproblems. To cope with large data scenarios, databases apply a similar Divide and Conquer strategy called *partitioning*. Partitioning divides a logical database or its components into independent partitions, each logically maintaining data with similar properties. This division simplifies data management, queries, and maintenance by breaking down large data sets into smaller, manageable parts.

7.8.1.2 Table Partitioning Technology

Table partitioning divides a large table or index into smaller, easier-to-manage logical units (called *partitions*), reducing the impact of table query, modification, and other statements. By using a partition key, users can quickly locate the partition containing the data they need, avoiding a full scan of the entire table. In addition, users can perform DDL and DML operations concurrently on different partitions. From the users' perspective, table partitioning technology offers the following capabilities:

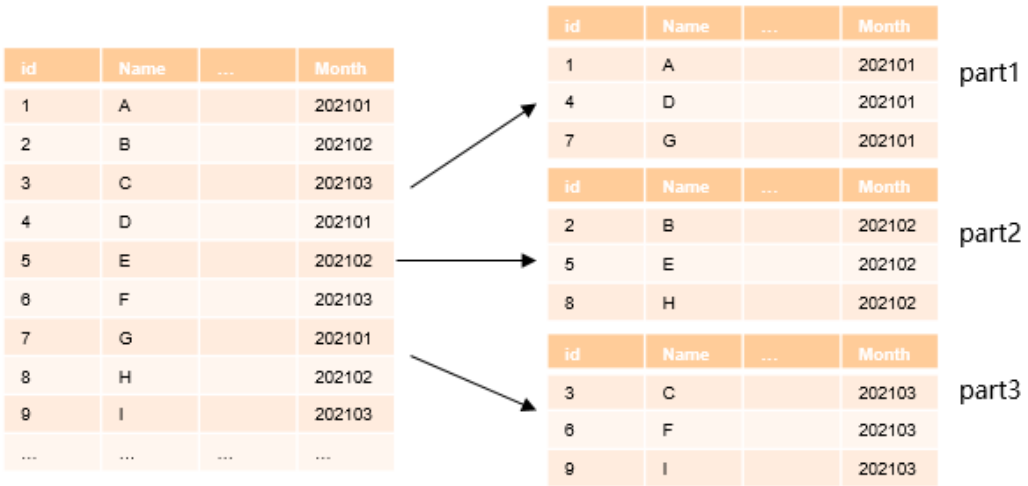
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Enhanced query efficiency in large data scenarios: The system logically partitions table data using a partition key. When executing a query, it accesses only the relevant partition subset instead of scanning the entire table. This partition pruning technology significantly enhances query performance and delivers performance gains by orders of magnitude.
- Reduced impact of concurrent O&M and query operations: Partitioning technology significantly reduces mutual interference between concurrent DML and DDL statements. This advantage is especially evident in large, time-partitioned data sets. For instance, the system can import new data partitions, run real-time point queries, and carry out O&M tasks such as cleaning or merging outdated partitions independently, without mutual interference.
- Flexible data O&M in large data scenarios: Partitioning technology physically isolates data into separate partitions. Each partition can be independently configured with physical properties, such as compression, physical storage, and tablespace. Moreover, partitioning allows you to perform data management tasks—such as loading data, creating or rebuilding indexes, and backing up or restoring data—at the partition level, avoiding operations on the entire table and significantly cutting operation time.

7.8.1.3 Optimized Data Queries by Partition

The performance benefits of partitioned tables in data queries are most evident in scenarios involving predicate queries on partition keys. For example, consider a table partitioned by month, as shown in **Figure 7-5**.

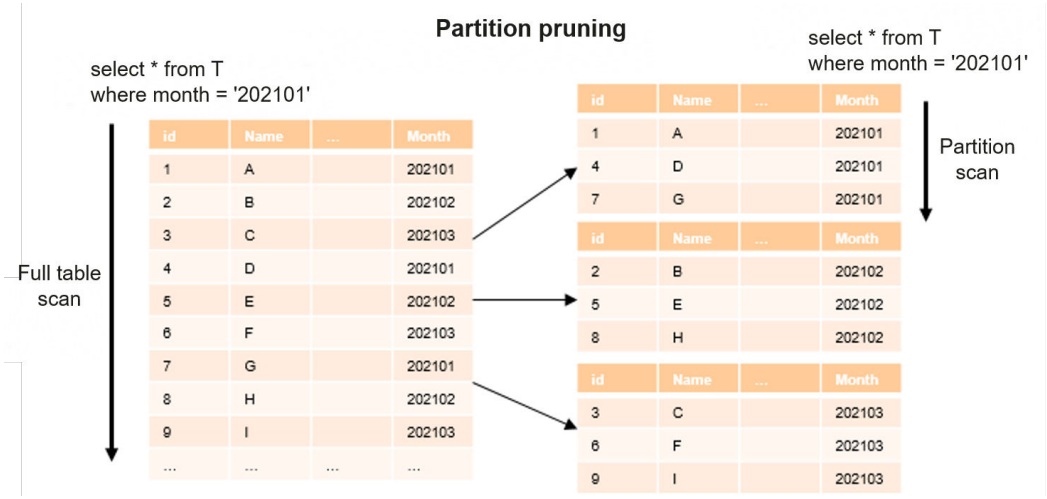
Figure 7-5 Example of a partitioned table



Designing the table as a regular table forces queries to perform full table scans. You can avoid this by redesigning the table to use month as the partition key, so queries scan only the relevant partitions. When the table stores large volumes of data over extended periods, reducing the scanned data significantly improves performance, as shown in **Figure 7-6**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-6 Example of partition pruning



7.8.1.4 Partition O&M

Data lifecycle management (DLM) encompasses processes and policies that govern data throughout its entire lifecycle. A key DLM task is to assign the most suitable and cost-effective storage media for data at different stages. Specifically, it stores frequently accessed new data on high-speed, highly available storage, while migrating older, infrequently accessed data to lower-cost storage with slightly reduced performance. Since old data is rarely updated, compressing it and converting it to read-only storage is a practical approach.

Partitioned tables create ideal conditions for implementing the DLM solution. By assigning different tablespaces to each partition, the system maximizes usability and optimizes cost-effectiveness throughout the data lifecycle. While database O&M personnel handle the underlying configuration on the server, end users can query tables as usual without noticing any changes. In addition, each partition supports independent O&M operations such as backup, restoration, and reindex. This allows separate management of different data subsets to meet the diverse requirements of users' service scenarios.

7.8.2 Introduction to Partitioned Tables

A partitioned table is a table whose data is logically divided into partitions based on partition keys and partitioning policies. Essentially, this is a horizontal partitioning approach. Using partitioned tables helps enhance the performance, manageability, and availability of database applications, and reduce the total cost of ownership (TCO) for storing large volumes of data. Partitioning divides tables, indexes, and index-organized tables into smaller units for refined management and access.

GaussDB provides various partitioning policies and extensions to meet the requirements of different service scenarios. Partitioning policies are completely implemented inside databases; users are unaware of them. Once implemented, these policies allow data to migrate smoothly without requiring any changes to user applications. This section covers the fundamental concepts of GaussDB's partitioned tables from the following perspectives:

- Partitioned tables: the catalog storage mode and underlying internal principles of partitioned tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Partitioning policies: the features, optimization, and effects delivered by basic partitioning types

7.8.2.1 Basic Concepts

7.8.2.1.1 Partitioned Table

A partitioned table serves as a table object that users directly operate on, supporting standard DML actions such as insert, delete, select, and update. Typically, users define a partitioned table by explicitly including the PARTITION BY clause in the DDL statement CREATE TABLE. Once a partitioned table is created, a record is added to the pg_class system catalog to represent it. This record's **parttype** is either **'p'** (indicating a partition) or **'s'** (indicating a subpartition).

Note that a partitioned table is a logical construct; the corresponding physical table does not actually store any data.

Example 1: **t1_hash** is a hash partitioned table.

```
gaussdb=# CREATE TABLE t1_hash (c1 INT, c2 INT, c3 INT)
PARTITION BY HASH(c1)
(
   PARTITION p0,
   PARTITION p1,
   PARTITION p2,
   PARTITION p3,
   PARTITION p4,
   PARTITION p5,
   PARTITION p6,
   PARTITION p7,
   PARTITION p8,
   PARTITION p9
);

gaussdb=# \d+ t1_hash
                  Table "public.t1_hash"
Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
c1     | integer |           | plain   |              |
c2     | integer |           | plain   |              |
c3     | integer |           | plain   |              |
Partition By HASH(c1)
Number of partitions: 10 (View pg_partition to check each partition range.)
Distribute By: HASH(c1)
Location Nodes: ALL DATANODES
Has OIDs: no
Options: orientation=row, compression=no

-- Query the partitioning type of t1_hash.
gaussdb=# SELECT relname, parttype FROM pg_class WHERE relname = 't1_hash';
relname | parttype
---------+----------
t1_hash | p
(1 row)

-- Drop t1_hash.
gaussdb=# DROP TABLE t1_hash;
```

Example 2: **t1_sub_rr** is a range-list subpartitioned table.

```
gaussdb=# CREATE TABLE t1_sub_rr (
   c1 INT,
   c2 INT,
   c3 INT
)
PARTITION BY RANGE (c1)
SUBPARTITION BY LIST (c2)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(
    PARTITION p_2021 VALUES LESS THAN (2022) (
        SUBPARTITION p_2021_1 VALUES (1),
        SUBPARTITION p_2021_2 VALUES (2),
        SUBPARTITION p_2021_3 VALUES (3)
    ),
    PARTITION p_2022 VALUES LESS THAN (2023) (
        SUBPARTITION p_2022_1 VALUES (1),
        SUBPARTITION p_2022_2 VALUES (2),
        SUBPARTITION p_2022_3 VALUES (3)
    ),
    PARTITION p_2023 VALUES LESS THAN (2024) (
        SUBPARTITION p_2023_1 VALUES (1),
        SUBPARTITION p_2023_2 VALUES (2),
        SUBPARTITION p_2023_3 VALUES (3)
    ),
    PARTITION p_2024 VALUES LESS THAN (2025) (
        SUBPARTITION p_2024_1 VALUES (1),
        SUBPARTITION p_2024_2 VALUES (2),
        SUBPARTITION p_2024_3 VALUES (3)
    ),
    PARTITION p_2025 VALUES LESS THAN (2026) (
        SUBPARTITION p_2025_1 VALUES (1),
        SUBPARTITION p_2025_2 VALUES (2),
        SUBPARTITION p_2025_3 VALUES (3)
    ),
    PARTITION p_2026 VALUES LESS THAN (2027) (
        SUBPARTITION p_2026_1 VALUES (1),
        SUBPARTITION p_2026_2 VALUES (2),
        SUBPARTITION p_2026_3 VALUES (3)
    )
);

gaussdb=# \d+ t1_sub_rr
                    Table "public.t1_sub_rr"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
 c3     | integer |           | plain   |              |
Partition By RANGE(c1) Subpartition By LIST(c2)
Number of partitions: 6 (View pg_partition to check each partition range.)
Number of subpartitions: 18 (View pg_partition to check each subpartition range.)
Has OIDs: no
Distribute By: HASH(c1)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

-- Query the partitioning type of t1_sub_rr.
gaussdb=# SELECT relname, parttype FROM pg_class WHERE relname = 't1_sub_rr';
  relname  | parttype
-----------+----------
 t1_sub_rr | s
(1 row)

-- Drop t1_sub_rr.
gaussdb=# DROP TABLE t1_sub_rr;
```

7.8.2.1.2 Partition

A partition within a partitioned table is a table that actually stores data. Once partitions are created, records are added to pg_partition. Using **parentid** as the foreign key, these partitions are joined with partitioned table OIDs in the pg_class system catalog.

Example: **t1_hash** is a partitioned table.

```
gaussdb=# CREATE TABLE t1_hash (c1 INT, c2 INT, c3 INT)
PARTITION BY HASH(c1)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(
    PARTITION p0,
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4,
    PARTITION p5,
    PARTITION p6,
    PARTITION p7,
    PARTITION p8,
    PARTITION p9
);

-- Query the partitioning type of t1_hash.
gaussdb=# SELECT oid, relname, parttype FROM pg_class WHERE relname = 't1_hash';
oid  | relname | parttype
-------+---------+----------
16685 | t1_hash | p
(1 row)

-- Query the partition information of t1_hash.
gaussdb=# SELECT oid, relname, parttype, parentid FROM pg_partition WHERE parentid = 16685;
oid  | relname | parttype | parentid
-------+---------+----------+----------
16688 | t1_hash | r        |    16685
16689 | p0      | p        |    16685
16690 | p1      | p        |    16685
16691 | p2      | p        |    16685
16692 | p3      | p        |    16685
16693 | p4      | p        |    16685
16694 | p5      | p        |    16685
16695 | p6      | p        |    16685
16696 | p7      | p        |    16685
16697 | p8      | p        |    16685
16698 | p9      | p        |    16685
(11 rows)

-- Drop t1_hash.
gaussdb=# DROP TABLE t1_hash;
```

7.8.2.1.3 Partition Key

A partition key consists of one or more columns. The system uses the partition key value and partitioning policy together to uniquely identify the partition that stores a specific tuple. Generally, the partition key value is specified in the PARTITION BY clause during table creation.

```
CREATE TABLE table_name (…) PARTITION BY part_strategy (partition_key) (…)
```

NOTICE

Range partitioned tables and list partitioned tables permits using up to 16 columns in the partition key, while other partitioned tables limit the partition key to a single column.

7.8.2.2 Partitioning Policies

Partitioning policies define how data maps to partition routes in a partitioned table. You specify these policies using the PARTITION BY clause in the DDL statement CREATE TABLE.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Common partitioning policies include range partitioning based on conditions, hash partitioning based on hash functions, and list partitioning based on data enumeration.

```
CREATE TABLE table_name (…) PARTITION BY partition_strategy (partition_key) (…)
```

7.8.2.2.1 Range Partitioning

Range partitioning maps data to partitions based on the partition key value range of each partition. This partitioning type is common in real-world production systems and is typically used for time-based data, such as dates or timestamps.

There are two syntax formats for range partitioning:

- VALUES LESS THAN syntax format

  If the VALUE LESS THAN clause is present in the syntax, the range partitioning policy permits using up to 16 columns as the partition key.

  - The following example shows how to use a single-column partition key:

    ```
    gaussdb=# CREATE TABLE range_sales
    (
       product_id      INT4 NOT NULL,
       customer_id     INT4 NOT NULL,
       time            DATE,
       channel_id      CHAR(1),
       type_id         INT4,
       quantity_sold   NUMERIC(3),
       amount_sold     NUMERIC(10,2)
    )
    PARTITION BY RANGE (time)
    (
       PARTITION date_202001 VALUES LESS THAN ('2020-02-01'),
       PARTITION date_202002 VALUES LESS THAN ('2020-03-01'),
       PARTITION date_202003 VALUES LESS THAN ('2020-04-01'),
       PARTITION date_202004 VALUES LESS THAN ('2020-05-01')
    );

    gaussdb=# DROP TABLE range_sales;
    ```

    date_202002 represents the partition for February 2020, storing the data generated from February 1, 2020 to February 29, 2020.

    Each partition includes a VALUES LESS clause that sets its upper limit. When the partition key value of a data item reaches or exceeds the upper limit of the current partition, the data item will be placed into the next partition. Except for the first partition, all other partitions have a lower limit implicitly specified by the VALUES LESS clause of the previous partition.

    You can assign the MAXVALUE keyword to the highest partition. MAXVALUE represents a virtually infinite value and ranks higher than any other partition key value, including null.

  - The following example shows how to use a multi-column partition key:

    ```
    gaussdb=# CREATE TABLE range_sales_with_multiple_keys
    (
       c1      INT4 NOT NULL,
       c2      INT4 NOT NULL,
       c3      CHAR(1)
    )
    PARTITION BY RANGE (c1,c2)
    (
       PARTITION p1 VALUES LESS THAN (10,10),
       PARTITION p2 VALUES LESS THAN (10,20),
       PARTITION p3 VALUES LESS THAN (20,10)
    );
    ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(9,5,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(9,20,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(9,21,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(10,5,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(10,15,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(10,20,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(10,21,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(11,5,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(11,20,'a');
gaussdb=# INSERT INTO range_sales_with_multiple_keys VALUES(11,21,'a');

gaussdb=# SELECT * FROM range_sales_with_multiple_keys PARTITION (p1);
 c1 | c2 | c3
----+----+----
  9 |  5 | a
  9 | 20 | a
  9 | 21 | a
 10 |  5 | a
(4 rows)

gaussdb=# SELECT * FROM range_sales_with_multiple_keys PARTITION (p2);
 c1 | c2 | c3
----+----+----
 10 | 15 | a
(1 row)

gaussdb=# SELECT * FROM range_sales_with_multiple_keys PARTITION (p3);
 c1 | c2 | c3
----+----+----
 10 | 20 | a
 10 | 21 | a
 11 |  5 | a
 11 | 20 | a
 11 | 21 | a
(5 rows)

gaussdb=# DROP TABLE range_sales_with_multiple_keys;
```

NOTE

With a multi-column partition key, the system follows these partitioning rules:

1. It starts by comparing the first column.
2. If the inserted value in the current column is less than the column's boundary value in the partition, the system inserts it directly.
3. If the inserted value equals the column's boundary value, the system compares the next column's inserted value with the next column's boundary value.
4. If the inserted value exceeds the column's boundary value, the system moves to the next partition for comparison.

- START END syntax format

  If the START END clause is present in the syntax, the range partitioning policy only permits using a single column as the partition key.

  Example:

```
-- Create tablespaces.
gaussdb=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';
gaussdb=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';
gaussdb=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';
gaussdb=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';
-- Create a temporary schema.
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# SET CURRENT_SCHEMA TO tpcds;
-- Create a partitioned table with a partition key of the integer type.
gaussdb=# CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
PARTITION BY RANGE (c2) (
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
    PARTITION p4 START(2500),
    PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
);
ENABLE ROW MOVEMENT;

-- View information about the partitioned table.
gaussdb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
    p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
relname | boundaries | spcname
-------------+------------+---------------
p1_0 | {1} | startend_tbs2
p1_1 | {201} | startend_tbs2
p1_2 | {401} | startend_tbs2
p1_3 | {601} | startend_tbs2
p1_4 | {801} | startend_tbs2
p1_5 | {1000} | startend_tbs2
p2 | {2000} | startend_tbs1
p3 | {2500} | startend_tbs3
p4 | {3000} | startend_tbs1
p5_1 | {4000} | startend_tbs4
p5_2 | {5000} | startend_tbs4
startend_pt | | startend_tbs1
(12 rows)

-- Drop the table and schema.
gaussdb=# DROP TABLE tpcds.startend_pt;
DROP TABLE
gaussdb=# DROP SCHEMA tpcds;
DROP SCHEMA
```

7.8.2.2.2 Hash Partitioning

Hash partitioning maps data to partitions based on GaussDB's hash algorithm and partition keys. If the partition keys' value ranges are free of data skew, the hash algorithm evenly distributes data rows across partitions, keeping partition sizes roughly equal. This is an ideal method for achieving even data distribution among partitions. Hash partitioning is also an easy-to-use alternative to range partitioning, especially when the data to be partitioned is not historical data or has no obvious partition key. For example:

```
gaussdb=# CREATE TABLE bmsql_order_line (
    ol_w_id          INTEGER   NOT NULL,
    ol_d_id          INTEGER   NOT NULL,
    ol_o_id          INTEGER   NOT NULL,
    ol_number        INTEGER   NOT NULL,
    ol_i_id          INTEGER   NOT NULL,
    ol_delivery_d    TIMESTAMP,
    ol_amount        DECIMAL(6,2),
    ol_supply_w_id   INTEGER,
    ol_quantity      INTEGER,
    ol_dist_info     CHAR(24)
)
-- Predefine 100 partitions.
PARTITION BY HASH(ol_d_id)
(
    PARTITION p0,
    PARTITION p1,
    PARTITION p2,
    …
    PARTITION p99
);

-- Drop the table.
gaussdb=# DROP TABLE bmsql_order_line;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

In the preceding example, the **bmsql_order_line** table is partitioned based on the **ol_d_id** column. Since this column serves as an identifier without time characteristics, it cannot distinguish data by a specific dimension. In this situation, the hash partitioning policy turns out to be an ideal choice.

Compared to other partitioning policies, users create hash partitions simply by specifying the partition key and the number of partitions, while ensuring the partition key does not cause serious data skew (where one or more values appear with extremely high frequency). Hash partitioning evenly distributes data across all partitions, significantly enhancing the usability of partitioned tables.

7.8.2.2.3 List Partitioning

List partitioning allows users to specify a list of discrete values for the partition key in the description of each partition to control how data rows map to partitions. The advantage of this partitioning method is that it partitions data in the form of enumerated values, making it especially suitable for grouping and organizing unordered or unrelated data sets.

If any values are not defined in the list, the system stores them in the **DEFAULT** partition instead of reporting an error even though they cannot map to any partition. For example:

```
gaussdb=# CREATE TABLE bmsql_order_line (
    ol_w_id          INTEGER   NOT NULL,
    ol_d_id          INTEGER   NOT NULL,
    ol_o_id          INTEGER   NOT NULL,
    ol_number        INTEGER   NOT NULL,
    ol_i_id          INTEGER   NOT NULL,
    ol_delivery_d    TIMESTAMP,
    ol_amount        DECIMAL(6,2),
    ol_supply_w_id   INTEGER,
    ol_quantity      INTEGER,
    ol_dist_info     CHAR(24)
)
PARTITION BY LIST(ol_d_id)
(
    PARTITION p0 VALUES (1,4,7),
    PARTITION p1 VALUES (2,5,8),
    PARTITION p2 VALUES (3,6,9),
    PARTITION p3 VALUES (DEFAULT)
);

-- Drop the table.
gaussdb=# DROP TABLE bmsql_order_line;
```

The preceding example resembles the earlier hash partitioning example, with the table partitioned by the **ol_d_id** column. However, list partitioning directly defines the possible values of **ol_d_id**. Any data not included in the list goes into the **p3** partition (**DEFAULT** partition).

Compared to hash partitioning, list partitioning offers you greater control over partition keys and can accurately store target data in the intended partitions. However, if there are a large number of values in the list, the partition definition can become cumbersome. In such cases, hash partitioning is recommended.

In general, list partitioning and hash partitioning are often used to group and organize unordered and unrelated data sets.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

List partitioning permits using up to 16 columns in the partition key. It applies distinct rules for handling enumerated NULL values depending on whether the partition key is single-column or multi-column. When you define a single-column partition key, the list of enumerated values cannot include NULL. However, when you define a multi-column partition key, the list can include NULL.

7.8.2.2.4 Subpartitioning

Subpartitioning, also known as composite partitioning, combines different partitioning policies. It first partitions a table using one policy, and then further divides each partition into subpartitions using a different policy. All subpartitions in a given partition represent a logical subset of data. Common types of composite partitioning include:

- Range-range
- Range-list
- Range-hash
- List-range
- List-list
- List-hash
- Hash-range
- Hash-list
- Hash-hash

Example:

```
-- Range-range
gaussdb=# CREATE TABLE t_range_range (
    c1 INT,
    c2 INT,
    c3 INT
)
PARTITION BY RANGE (c1)
SUBPARTITION BY RANGE (c2)
(
    PARTITION p1 VALUES LESS THAN (10) (
        SUBPARTITION p1sp1 VALUES LESS THAN (5),
        SUBPARTITION p1sp2 VALUES LESS THAN (10)
    ),
    PARTITION p2 VALUES LESS THAN (20) (
        SUBPARTITION p2sp1 VALUES LESS THAN (15),
        SUBPARTITION p2sp2 VALUES LESS THAN (20)
    )
);
gaussdb=# DROP TABLE t_range_range;

-- Range-list
gaussdb=# CREATE TABLE t_range_list (
    c1 INT,
    c2 INT,
    c3 INT
)
PARTITION BY RANGE (c1)
SUBPARTITION BY LIST (c2)
(
    PARTITION p1 VALUES LESS THAN (10) (
        SUBPARTITION p1sp1 VALUES (1, 2),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    SUBPARTITION p1sp2 VALUES (3, 4)
  ),
  PARTITION p2 VALUES LESS THAN (20) (
    SUBPARTITION p2sp1 VALUES (1, 2),
    SUBPARTITION p2sp2 VALUES (3, 4)
  )
);
gaussdb=# DROP TABLE t_range_list;

-- Range-hash
gaussdb=# CREATE TABLE t_range_hash (
  c1 INT,
  c2 INT,
  c3 INT
)
PARTITION BY RANGE (c1)
SUBPARTITION BY HASH (c2)
SUBPARTITIONS 2
(
  PARTITION p1 VALUES LESS THAN (10),
  PARTITION p2 VALUES LESS THAN (20)
);
gaussdb=# DROP TABLE t_range_hash;

-- List-range
gaussdb=# CREATE TABLE t_list_range (
  c1 INT,
  c2 INT,
  c3 INT
)
PARTITION BY LIST (c1)
SUBPARTITION BY RANGE (c2)
(
  PARTITION p1 VALUES (1, 2) (
    SUBPARTITION p1sp1 VALUES LESS THAN (5),
    SUBPARTITION p1sp2 VALUES LESS THAN (10)
  ),
  PARTITION p2 VALUES (3, 4) (
    SUBPARTITION p2sp1 VALUES LESS THAN (5),
    SUBPARTITION p2sp2 VALUES LESS THAN (10)
  )
);
gaussdb=# DROP TABLE t_list_range;

-- List-list
gaussdb=# CREATE TABLE t_list_list (
  c1 INT,
  c2 INT,
  c3 INT
)
PARTITION BY LIST (c1)
SUBPARTITION BY LIST (c2)
(
  PARTITION p1 VALUES (1, 2) (
    SUBPARTITION p1sp1 VALUES (1, 2),
    SUBPARTITION p1sp2 VALUES (3, 4)
  ),
  PARTITION p2 VALUES (3, 4) (
    SUBPARTITION p2sp1 VALUES (1, 2),
    SUBPARTITION p2sp2 VALUES (3, 4)
  )
);
gaussdb=# DROP TABLE t_list_list;

-- List-hash
gaussdb=# CREATE TABLE t_list_hash (
  c1 INT,
  c2 INT,
  c3 INT
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
)
PARTITION BY LIST (c1)
SUBPARTITION BY HASH (c2)
SUBPARTITIONS 2
(
    PARTITION p1 VALUES (1, 2),
    PARTITION p2 VALUES (3, 4)
);
gaussdb=# DROP TABLE t_list_hash;

-- Hash-range
gaussdb=# CREATE TABLE t_hash_range (
    c1 INT,
    c2 INT,
    c3 INT
)
PARTITION BY HASH (c1)
PARTITIONS 2
SUBPARTITION BY RANGE (c2)
(
    PARTITION p1 (
        SUBPARTITION p1sp1 VALUES LESS THAN (5),
        SUBPARTITION p1sp2 VALUES LESS THAN (10)
    ),
    PARTITION p2 (
        SUBPARTITION p2sp1 VALUES LESS THAN (5),
        SUBPARTITION p2sp2 VALUES LESS THAN (10)
    )
);
gaussdb=# DROP TABLE t_hash_range;

-- Hash-list
gaussdb=# CREATE TABLE t_hash_list (
    c1 INT,
    c2 INT,
    c3 INT
)
PARTITION BY HASH (c1)
PARTITIONS 2
SUBPARTITION BY LIST (c2)
(
    PARTITION p1 (
        SUBPARTITION p1sp1 VALUES (1, 2),
        SUBPARTITION p1sp2 VALUES (3, 4)
    ),
    PARTITION p2 (
        SUBPARTITION p2sp1 VALUES (1, 2),
        SUBPARTITION p2sp2 VALUES (3, 4)
    )
);
gaussdb=# DROP TABLE t_hash_list;

-- Hash-hash
gaussdb=# CREATE TABLE t_hash_hash (
    c1 INT,
    c2 INT,
    c3 INT
)
PARTITION BY HASH (c1)
PARTITIONS 2
SUBPARTITION BY HASH (c2)
SUBPARTITIONS 2
(
    PARTITION p1,
    PARTITION p2
);
gaussdb=# DROP TABLE t_hash_hash;

-- List-list
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code, user_no)
(
  PARTITION p_201901 VALUES ( '201902' ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES (('1','120')),
    SUBPARTITION p_201902_b VALUES (('2','240'))
  )
);
gaussdb=# SELECT * FROM pg_get_tabledef('list_list');
gaussdb=# INSERT INTO list_list VALUES('201902', '1', '120', 1);
gaussdb=# INSERT INTO list_list VALUES('201902', '2', '240', 1);
gaussdb=# INSERT INTO list_list VALUES('201902', '1', '120', 1);
gaussdb=# INSERT INTO list_list VALUES('201903', '2', '240', 1);
gaussdb=# INSERT INTO list_list VALUES('201903', '1', '120', 1);
gaussdb=# INSERT INTO list_list VALUES('201903', '2', '240', 1);
gaussdb=# SELECT * FROM list_list ORDER BY month_code;
gaussdb=# DROP TABLE list_list;

-- Hash-range
gaussdb=# CREATE TABLE hash_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY range (dept_code,user_no)
(
  PARTITION p_201901,
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a VALUES LESS THAN ( '2' ,'2'),
    SUBPARTITION p_201902_b VALUES LESS THAN ( '3' ,'4')
  )
);
gaussdb=# INSERT INTO hash_range VALUES('201901', '1', '1', 1);
gaussdb=# INSERT INTO hash_range VALUES('201901', '2', '1', 1);
gaussdb=# INSERT INTO hash_range VALUES('201901', '1', '1', 1);
gaussdb=# INSERT INTO hash_range VALUES('201903', '2', '1', 1);
gaussdb=# INSERT INTO hash_range VALUES('201903', '1', '1', 1);
gaussdb=# INSERT INTO hash_range VALUES('201903', '2', '1', 1);
gaussdb=# SELECT * FROM hash_range;
gaussdb=# DROP TABLE hash_range;
```

NOTE

Interval partitioning can be viewed as a special form of range partitioning.

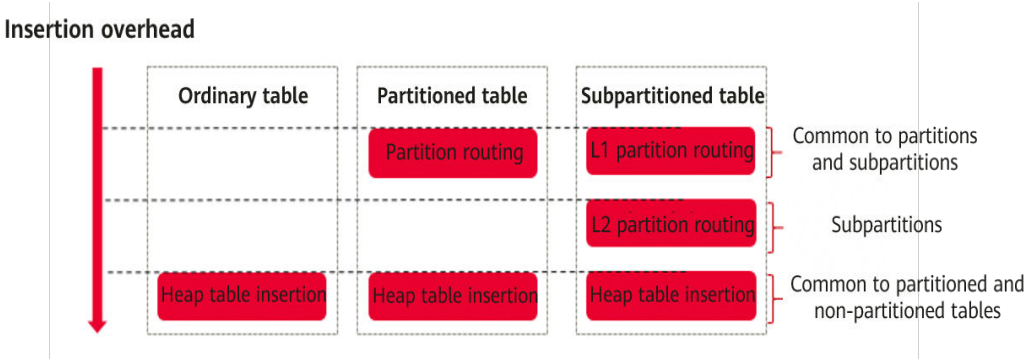
7.8.2.2.5 Impact on Import Performance

In GaussDB, inserting data into a partitioned table incurs higher overhead than inserting into a non-partitioned table due to the added step of partition routing.

As illustrated in **Figure 7-7**, the data insertion overhead of partitioned tables consists of two parts: (1) Heap table insertion, which solves the problem of inserting tuples into the appropriate heap table. This insertion is common to both ordinary and partitioned tables. (2) Partition routing, which solves the problem of inserting tuples into the appropriate partitioned tables (**partRel**).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-7 Inserting data into ordinary tables vs. partitioned tables



Therefore, data insertion is mainly optimized from the following aspects:

- Heap table insertion:
  - Optimized noise floor for operators
  - Heap data insertion
  - Optimized index build
- Partition routing:
  - Optimized logic for the routing algorithm
  - Optimized noise floor for routing, including enabling the **partRel** handle of partitioned tables and adding the logic overhead of function calls

  NOTE

  The performance of partition routing becomes especially noticeable when you execute a single INSERT statement with a large volume of data. In contrast, the UPDATE process is more complex: it first locates the tuple to be updated, then deletes it, and finally inserts a new tuple. Compared to a single INSERT statement, UPDATE involves more steps and greater complexity. As a result, partition routing performs less efficiently during UPDATE than during a single INSERT statement.

Table 7-7 explains the routing algorithm's logic under different partitioning types.

Table 7-7 Routing algorithm's logic

Partitioning Type	Routing Complexity	Implementation
Range partitioning	O(logN)	Binary search
Hash partitioning	O(1)	Key-partoid hash table
List partitioning	O(1)	Key-partoid hash table

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

Partition routing primarily determines each imported tuple's partition by calculating its partition key. This step incurs additional overhead compared to non-partitioned tables. The impact on final data import performance depends on factors such as server CPU capabilities, table width, and actual disk and memory capacities.

Generally, the performance impact can be roughly estimated as follows:

- On x86 servers, importing data into a partitioned table runs slightly slower (< 10%) than importing into an ordinary table.
- On Arm servers, data import performance reduces by 20%.

The slight performance gap between x86 and Arm servers stems from the fact that partition routing relies on intensive in-memory computations. Mainstream x86 CPUs offer marginally better single-core instruction processing than their Arm counterparts.

7.8.2.3 Basic Usage of Partitions

7.8.2.3.1 Creating Partitioned Tables

Specifications and Limitations

Each table can accommodate a maximum of 1,048,575 partitions and subpartitions in total. Excessive partitions may result in an increased number of system files, consuming more system cache and lock resources, which degrades system performance. In the default setup, it is advisable to keep the number of partitions per table at 400 or less. If the number of hot partitions in use exceeds 400, adjust the value of **FASTPATH_PART** (refer to the GUC parameter **num_internal_lock_partitions** for the adjustment method). After adjustment, ensure that the number of partitions in a single table does not exceed 2,000.

Creating a Partitioned Table

SQL offers powerful and flexible functionalities, with a complex syntax tree—a characteristic that also applies to partitioned tables. Creating a partitioned table can be viewed as adding partitioning attributes to a non-partitioned table. Therefore, the syntax for creating partitioned tables can be considered an extension of the standard CREATE TABLE statement used for non-partitioned tables. Specifically, it introduces the PARTITION BY clause and requires specifying the following three core partitioning elements:

- **partType**: describes the partitioning policy employed by the partitioned table. Supported policies include range partitioning, list partitioning, and hash partitioning.
- **partKey**: defines the columns used to partition the table. Currently, range partitioning and list partitioning permit using multiple columns (up to 16) as the partition key, whereas hash partitioning is limited to a single column.
- **partExpr**: describes the partitioning mode of the table, that is, the mapping between key values and partitions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

These three elements are explicitly defined in the PARTITION BY clause of the CREATE TABLE statement, in the PARTITION BY partType (partKey) (partExpr[,partExpr]…) format. Below is the basic syntax framework:

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
(
   [ /* This part is inherited from the CREATE TABLE statement for ordinary tables. */
   { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }[, ... ]
   ]
)
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
/* Range partitioning scenario. */
PARTITION BY RANGE (partKey) (
   { partition_start_end_item [, ... ] | partition_less_then_item [, ... ] }
)
/* List partitioning scenario. */
PARTITION BY LIST (partKey)
(
   PARTITION partition_name VALUES (list_values_clause) [ TABLESPACE tablespace_name [, ... ] ]
...
)
/* Hash partitioning scenario. */
PARTITION BY HASH (partKey) (
   PARTITION partition_name [ TABLESPACE tablespace_name [, ... ] ]
...
)
/* Enable or disable row movement for the partitioned table. */
[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

In non-multi-primary mode, the use of partitioned tables is subject to the following specifications and limitations:

- Range partitioning and list partitioning permit using up to 16 columns as the partition key, while hash partitioning is limited to a single column.
- Null values must not be inserted into partition keys, except for hash partitioning. Otherwise, DML statements will result in an error. However, null values can be inserted when the **MAXVALUE** partition is defined for a range partitioned table or the **DEFAULT** partition is defined for a list partitioned table.

Creating a Subpartitioned Table

A subpartitioned table can be considered a further extension of a partitioned table. In the structure of a subpartitioned table, each partition acts as a logical table, which does not store data directly; instead, data is stored within subpartitions.

Essentially, a subpartitioned table is implemented by nesting two partitions together. For details on supported partitioning policies, see "SQL Reference > SQL Syntax > C > CREATE TABLE PARTITION" in *Reference*.

Commonly used subpartitioning combinations include range-range, range-list, range-hash, list-range, list-list, list-hash, hash-range, hash-list, and hash-hash. Currently, subpartitioning is only supported for row-store tables. Below is the basic syntax framework of subpartitioning:

```
CREATE TABLE [ IF NOT EXISTS ] subpartition_table_name
(
   [ /* This part is inherited from the CREATE TABLE statement for ordinary tables. */
   { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   | table_constraint
   | LIKE source_table [ like_option [...] ] } [, ... ]
   ]
)
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
PARTITION BY {RANGE | LIST | HASH} (partKey) SUBPARTITION BY {RANGE | LIST | HASH} (partKey)
(
   PARTITION partition_name partExpr…  /* Partition */
   (
      SUBPARTITION partition_name partExpr … /* Subpartition */
      SUBPARTITION partition_name partExpr … /* Subpartition */
   ),
   PARTITION partition_name partExpr…  /* Partition */
   (
      SUBPARTITION partition_name partExpr … /* Subpartition */
      SUBPARTITION partition_name partExpr … /* Subpartition */
   ),
   …
)
[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

Specifications and limitations:

A subpartitioned table is constructed by combining any two of the following partition types: list, hash, and range.

Modifying Partition Attributes

Similar to non-partitioned tables, you can run the **ALTER TABLE** command to modify attributes for partitioned tables and partitions. Common modification statements include:

- ADD PARTITION
- DROP PARTITION
- TRUNCATE PARTITION
- SPLIT PARTITION
- MERGE PARTITION
- MOVE PARTITION
- EXCHANGE PARTITION
- RENAME PARTITION

The preceding statements are extensions of the ALTER TABLE statement for ordinary tables. Their usage in partitioned tables resembles that in ordinary tables. Below is the basic syntax framework for modifying partition attributes:

```
/* Basic ALTER TABLE syntax. */
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
action [, ... ];
```

For detailed usage of the ALTER TABLE statement in partitioned tables, refer to **Partitioned Table O&M** as well as "SQL Reference > SQL Syntax > A > ALTER TABLE PARTITION" and "SQL Reference > SQL Syntax > A > ALTER TABLE SUBPARTITION" in *Reference*.

7.8.2.3.2 Using and Managing Partitioned Tables

Partitioned tables support most of the same features as non-partitioned tables. For details, refer to the syntax for ordinary tables in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Not only this, partitioned tables support a wide range of partition-level operations, including DQL/DML statements (such as SELECT, INSERT, UPDATE, DELETE, UPSERT, and MERGE INTO), DDL statements (such as ADD, DROP, TRUNCATE, EXCHANGE, SPLIT, MERGE, MOVE, and RENAME), VACUUM/ANALYZE, and sparsely partitioned indexes.

A partition-level operation is generally performed by specifying a partition name or value. For example:

```
sql_action [ t_name ] { PARTITION | SUBPARTITION } { p_name | (p_name) };
sql_action [ t_name ] { PARTITION | SUBPARTITION } FOR (p_value);
```

You can target a specific partition by specifying either a partition name (**p_name**) or a partition value (**p_value**). This directs the operation to that partition only, without affecting any others. When you specify a partition name (**p_name**), the database attempts to locate the corresponding partition and throws an exception if the partition does not exist. When you specify a partition value (**p_value**), the database identifies the partition that contains the specified value.

For example, define a partitioned table like this:

```
gaussdb=# CREATE TABLE list_01
(
   id   INT,
   role VARCHAR(100),
   data VARCHAR(100)
)
PARTITION BY LIST (id)
(
   PARTITION p_list_1 VALUES(0,1,2,3,4),
   PARTITION p_list_2 VALUES(5,6,7,8,9),
   PARTITION p_list_3 VALUES(DEFAULT)
);

-- Drop the table.
gaussdb=# DROP TABLE list_01;
```

When specifying a partition, PARTITION p_list_1 is equivalent to PARTITION FOR (4), as both point to the same partition. Similarly, PARTITION p_list_3 is equivalent to PARTITION FOR (12), and they also represent the same partition.

7.8.2.3.3 DQL/DML on a Partitioned Table

Partitioning is implemented in the database kernel. Therefore, the syntax of DQL/DML operations on partitioned tables is the same as that on non-partitioned tables.

To make partitioned tables easy to use, GaussDB supports DQL/DML operations on specified partitions—you can specify PARTITION (*partname*) or PARTITION FOR (*partvalue*). Subpartition can also be specified using SUBPARTITION(*subpartname*) or SUBPARTITION FOR (*subpartvalue*).

When you perform DQL/DML operations on a specified partition, the system reports an error if the inserted data lies outside that partition. During queries, the database skips any data that does not belong to the target partition.

The following DQL/DML operations can be performed on a specified partition:

- SELECT
- INSERT
- UPDATE

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- DELETE
- UPSERT
- MERGE INTO

Here is an example of performing DQL/DML operations on a specified partition:

```
-- Create a partitioned table named list_02.
gaussdb=# CREATE TABLE IF NOT EXISTS list_02
(
    id   INT,
    role VARCHAR(100),
    data VARCHAR(100)
)
PARTITION BY LIST (id)
(
    PARTITION p_list_2 VALUES(0,1,2,3,4,5,6,7,8,9),
    PARTITION p_list_3 VALUES(10,11,12,13,14,15,16,17,18,19),
    PARTITION p_list_4 VALUES( DEFAULT ),
    PARTITION p_list_5 VALUES(20,21,22,23,24,25,26,27,28,29),
    PARTITION p_list_6 VALUES(30,31,32,33,34,35,36,37,38,39),
    PARTITION p_list_7 VALUES(40,41,42,43,44,45,46,47,48,49)
) ENABLE ROW MOVEMENT;
-- Insert data.
gaussdb=# INSERT INTO list_02 VALUES(null, 'alice', 'alice data');
gaussdb=# INSERT INTO list_02 VALUES(2, null, 'bob data');
gaussdb=# INSERT INTO list_02 VALUES(null, null, 'peter data');

-- Query the specified partition.
-- Query all data in the partitioned table.
gaussdb=# SELECT * FROM list_02 ORDER BY data;
 id | role  |    data
----+-------+------------
    | alice | alice data
  2 |       | bob data
    |       | peter data
(3 rows)
-- Query data in the p_list_2 partition.
gaussdb=# SELECT * FROM list_02 PARTITION (p_list_2) ORDER BY data;
 id | role |   data
----+------+----------
  2 |      | bob data
(1 row)
-- Query data in the partition corresponding to (100), that is, partition p_list_4.
gaussdb=# SELECT * FROM list_02 PARTITION FOR (100) ORDER BY data;
 id | role  |    data
----+-------+------------
    | alice | alice data
    |       | peter data
(2 rows)

-- Perform IUD operations on the specified partition.
-- Delete all data from the p_list_5 partition.
gaussdb=# DELETE FROM list_02 PARTITION (p_list_5);
-- Insert data into the p_list_7 partition. An error is reported because the data violates the partition
constraints.
gaussdb=# INSERT INTO list_02 PARTITION (p_list_7) VALUES(null, 'cherry', 'cherry data');
ERROR:  inserted partition key does not map to the table partition
-- Update data in the partition to which the partition value 100 belongs, that is, partition p_list_4.
gaussdb=# UPDATE list_02 PARTITION FOR (100) SET data = '';

-- Perform UPSERT.
gaussdb=# INSERT INTO list_02 (id, role, data) VALUES (1, 'test', 'testdata') ON DUPLICATE KEY UPDATE
role = VALUES(role), data = VALUES(data);

-- Perform MERGE INTO.
gaussdb=# CREATE TABLE IF NOT EXISTS list_tmp
(
    id   INT,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    role VARCHAR(100),
    data VARCHAR(100)
)
PARTITION BY LIST (id)
(
    PARTITION p_list_2 VALUES(0,1,2,3,4,5,6,7,8,9),
    PARTITION p_list_3 VALUES(10,11,12,13,14,15,16,17,18,19),
    PARTITION p_list_4 VALUES( DEFAULT ),
    PARTITION p_list_5 VALUES(20,21,22,23,24,25,26,27,28,29),
    PARTITION p_list_6 VALUES(30,31,32,33,34,35,36,37,38,39),
    PARTITION p_list_7 VALUES(40,41,42,43,44,45,46,47,48,49)) ENABLE ROW MOVEMENT;

gaussdb=# MERGE INTO list_tmp target
USING list_02 source
ON (target.id = source.id)
WHEN MATCHED THEN
  UPDATE SET target.data = source.data,
             target.role = source.role
WHEN NOT MATCHED THEN
  INSERT (id, role, data)
  VALUES (source.id, source.role, source.data);

-- Drop tables.
gaussdb=# DROP TABLE list_02;
gaussdb=# DROP TABLE list_tmp;
```

7.8.3 Query Optimizations on Partitioned Tables

NOTE

This section assumes **explain_perf_mode** is set to **normal**.

7.8.3.1 Partition Pruning

GaussDB uses partition pruning to optimize queries on partitioned tables. Its SQL engine scans only the partitions relevant to the query conditions. This optimization is automatically triggered when the query conditions meet the pruning criteria.

Partition pruning can be classified into static pruning and dynamic pruning based on the pruning phase.

- Static pruning occurs during the optimizer phase. Before generating an execution plan, the database has identified which partitions to access.
- Dynamic pruning occurs during the executor phase, either at the start of execution or while it is in progress. At the time the database generates an execution plan, it only confirms that pruning conditions are met but does not yet identify which partitions to access. The specific pruning details are determined by the executor.

Partition pruning supports the following:

- Supported partitioning levels: partitioning and subpartitioning.
- Supported partitioning types: all types, including range partitioning, hash partitioning, and list partitioning.
- Supported expression types: comparison expressions (<, <=, =, >=, >), logical expressions, and array expressions. For hash and list partitioning, pruning is applicable only when the comparison expression uses equality (=).
- Supported condition types: constants, parameters, and functions (limited to IMMUTABLE and STABLE functions). Note that parameterized path pruning

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

does not support STABLE functions. Complex conditions, such as subqueries, are not supported.

- If the partition key type differs from the condition type (triggering type conversion), ensure mutual convertibility between the types.

It should be noted that partition pruning is triggered only during partitioned table page scans and local index scans. Global indexes do not incorporate the concept of partitioning and therefore do not require pruning. For global indexes, the system determines which partitions to access based on the index conditions. As a general guideline: Use local indexes when queries include conditions on the partition key. Use global indexes when queries do not have such conditions, although they introduce additional overhead during partition maintenance operations. The index type can be explicitly declared using the GLOBAL or LOCAL keyword during index creation.

7.8.3.1.1 Static Pruning for Partitioned Tables

When a partitioned table query statement uses retrieval conditions that specify constants in the partition key, the optimizer employs the retrieval conditions from operators like indexscan, bitmapindexscan, and indexonlyscan to prune partitions. These retrieval conditions must include at least one partition key column. In the case of a multi-column partition key, the retrieval conditions can include any of the partition key columns.

NOTE

- Static pruning supports only constants. Certain non-constant conditions are preprocessed prior to plan generation. For instance, when Cplans are generated based on parameters, constant inputs are passed to IMMUTABLE/STABLE functions, or NULL inputs are passed to STRICT functions, related conditions are converted into constants, enabling static pruning to be subsequently performed.
- To support partitioned table pruning, the system forcibly converts filter conditions on the partition key to the partition key's data type during plan generation. This conversion differs from the implicit type conversion rule. As a result, errors may occur when converting these conditions on partition keys, while similar conversions on non-partition keys proceed without error.

- Typical scenarios that support static pruning:
  - Comparison expressions in non-multi-primary mode

```
-- Create a partitioned table.
gaussdb=# CREATE TABLE t1 (c1 int, c2 int)
PARTITION BY RANGE (c1)
(
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(20),
    PARTITION p3 VALUES LESS THAN(MAXVALUE)
);
gaussdb=# SET max_datanode_for_plan = 1;

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = 1;
                      QUERY PLAN
-------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = 1

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = 1
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    Filter: (t1.c1 = 1)
    Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 < 1;
                     QUERY PLAN
------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 < 1

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 < 1
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 < 1)
     Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 > 11;
                     QUERY PLAN
------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 > 11

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 > 11
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: 2
     ->  Partitioned Seq Scan on public.t1
           Output: c1, c2
           Filter: (t1.c1 > 11)
           Selected Partitions:  2..3

(15 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 is NULL;
                     QUERY PLAN
------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 IS NULL

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 IS NULL
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 IS NULL)
     Selected Partitions:  3

(12 rows)
```

- Logical expressions in non-multi-primary mode:

```
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = 1 AND c2 = 2;
                        QUERY PLAN
-----------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = 1 AND c2 = 2

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = 1 AND c2 = 2
 Datanode Name: datanode1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  Partitioned Seq Scan on public.t1
    Output: c1, c2
    Filter: ((t1.c1 = 1) AND (t1.c2 = 2))
    Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = 1 OR c1 = 2;
                        QUERY PLAN
-----------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = 1 OR c1 = 2

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = 1 OR c1 = 2
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: ((t1.c1 = 1) OR (t1.c1 = 2))
     Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE NOT c1 = 1;
                      QUERY PLAN
-----------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE NOT c1 = 1

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE NOT c1 = 1
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: 3
     ->  Partitioned Seq Scan on public.t1
           Output: c1, c2
           Filter: (t1.c1 <> 1)
           Selected Partitions:  1..3

(15 rows)
```

- Array expressions in non-multi-primary mode:

```
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 IN (1, 2, 3);
                            QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = ANY ('{1,2,3}'::integer[]))
     Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = ALL(ARRAY[1,
2, 3]);
                            QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = ALL (ARRAY[1, 2, 3])
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = ALL (ARRAY[1, 2, 3])
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: 0
     ->  Partitioned Seq Scan on public.t1
           Output: c1, c2
           Filter: (t1.c1 = ALL ('{1,2,3}'::integer[]))
           Selected Partitions:  NONE

(15 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = ANY(ARRAY[1,
2, 3]);
                              QUERY PLAN
-----------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = ANY ('{1,2,3}'::integer[]))
     Selected Partitions:  1

(12 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 =
SOME(ARRAY[1, 2, 3]);
                              QUERY PLAN
-----------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = ANY (ARRAY[1, 2, 3])
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = ANY ('{1,2,3}'::integer[]))
     Selected Partitions:  1

(12 rows)
```

- Typical scenarios that do not support static pruning:

  Subquery expressions in non-multi-primary mode:

```
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 WHERE c1 = ALL(SELECT c2
FROM t1 WHERE c1 > 10);
                              QUERY PLAN
-----------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: public.t1.c1, public.t1.c2
   Node/s: All datanodes
   ->  Partition Iterator
         Output: public.t1.c1, public.t1.c2
         Iterations: 3
         ->  Partitioned Seq Scan on public.t1
               Output: public.t1.c1, public.t1.c2
               Distribute Key: public.t1.c1
               Filter: (SubPlan 1)
               Selected Partitions:  1..3
               SubPlan 1
                 ->  Materialize
                       Output: public.t1.c2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                  ->  Streaming(type: BROADCAST)
                        Output: public.t1.c2
                        Spawn on: All datanodes
                        Consumer Nodes: All datanodes
                        ->  Partition Iterator
                              Output: public.t1.c2
                              Iterations: 2
                              ->  Partitioned Seq Scan on public.t1
                                    Output: public.t1.c2
                                    Distribute Key: public.t1.c1
                                    Filter: (public.t1.c1 > 10)
                                    Selected Partitions:  2..3
(26 rows)

-- Drop the table.
gaussdb=# DROP TABLE t1;
```

7.8.3.1.2 Dynamic Pruning for Partitioned Tables

When a partitioned table query statement includes variables in its retrieval conditions, the optimizer cannot access the user-provided bound parameters. Therefore, it parses only the retrieval conditions from operators like indexscan, bitmapindexscan, and indexonlyscan. Partition filtering completes only after the bound parameters are obtained during the subsequent executor phase.

These retrieval conditions must include at least one partition key column. In the case of a multi-column partition key, the retrieval conditions can include any of the partition key columns.

Currently, dynamic pruning is supported only in the prepare-bind-execute (PBE) and parameterized path scenarios.

Dynamic PBE Pruning

For partitioned table query statements where the partitioning key in retrieval conditions contains parameters, the system can only confirm that "partition pruning conditions are met" during the optimizer phase. Dynamic pruning occurs after execution starts and parameters are bound. The trigger behavior of dynamic pruning is the same as that of static pruning, differing only in the trigger phase.

NOTICE

- For certain STABLE functions, such as the type conversion function to_timestamp, different GUC parameter settings may lead to different pruning results. To optimize performance, you can analyze the table to regenerate a generic plan (Gplan).
- Dynamic PBE pruning relies on Gplans. Therefore, when evaluating whether a statement is eligible for dynamic PBE pruning, it is advisable to set **plan_cache_mode** to **'force_generic_plan'** to prevent interference from custom plans (Cplans).
- When dynamic pruning is used, the system cannot generate a sparsely partitioned index plan, because it has not identified the executing partition during plan generation.

- Typical scenarios that support dynamic PBE pruning:
  - Comparison expressions in non-multi-primary mode:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a partitioned table.
gaussdb=#CREATE TABLE t1 (c1 int, c2 int)
PARTITION BY RANGE (c1)
(
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(20),
    PARTITION p3 VALUES LESS THAN(MAXVALUE)
);

gaussdb=# PREPARE p1(int) AS SELECT * FROM t1 WHERE c1 = $1;
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p1(1);
                        QUERY PLAN
------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Node expr: $1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = $1

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = $1
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = $1)
     Selected Partitions:  1 (pbe-pruning)

(13 rows)
```

- Logical expressions in non-multi-primary mode:

```
gaussdb=# PREPARE p2(INT, INT) AS SELECT * FROM t1 WHERE c1 = $1 AND c2 = $2;
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p2(1, 2);
                            QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Node expr: $1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = $1 AND c2 = $2

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = $1 AND c2 = $2
 Datanode Name: datanode1
   Partitioned Seq Scan on public.t1
     Output: c1, c2
     Filter: ((t1.c1 = $1) AND (t1.c2 = $2))
     Selected Partitions:  1 (pbe-pruning)

(13 rows)
```

- Type conversions triggering implicit conversions in non-multi-primary mode:

```
gaussdb=# set plan_cache_mode = 'force_generic_plan';
gaussdb=# PREPARE p3(TEXT) AS SELECT * FROM t1 WHERE c1 = $1;
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p3('12');
                          QUERY PLAN
---------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Node expr: $1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1 = $1::bigint

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = $1::bigint
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: PART
     ->  Partitioned Seq Scan on public.t1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        Output: c1, c2
        Filter: (t1.c1 = ($1)::bigint)
        Selected Partitions:  2 (pbe-pruning)

(16 rows)
```

- Typical scenarios that do not support dynamic PBE pruning:
  - Subquery expressions in non-multi-primary mode:

```
gaussdb=# PREPARE p4(INT) AS SELECT * FROM t1 WHERE c1 = ALL(SELECT c2 FROM t1
WHERE c1 > $1);
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p4(1);
                          QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: public.t1.c1, public.t1.c2
   Node/s: All datanodes
   ->  Partition Iterator
         Output: public.t1.c1, public.t1.c2
         Iterations: 3
         ->  Partitioned Seq Scan on public.t1
               Output: public.t1.c1, public.t1.c2
               Distribute Key: public.t1.c1
               Filter: (SubPlan 1)
               Selected Partitions:  1..3
               SubPlan 1
                 ->  Materialize
                       Output: public.t1.c2
                       ->  Streaming(type: BROADCAST)
                             Output: public.t1.c2
                             Spawn on: All datanodes
                             Consumer Nodes: All datanodes
                             ->  Partition Iterator
                                   Output: public.t1.c2
                                   Iterations: PART
                                   ->  Partitioned Seq Scan on public.t1
                                         Output: public.t1.c2
                                         Distribute Key: public.t1.c1
                                         Filter: (public.t1.c1 > 1)
                                         Selected Partitions:  1..3
(26 rows)
```

  - Type conversions unable to directly trigger implicit conversions in non-multi-primary mode:

```
gaussdb=# PREPARE p5(name) AS SELECT * FROM t1 WHERE c1 = $1;
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p5('12');
                          QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE c1::text = '12'::text

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1::text = '12'::text
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: 3
     ->  Partitioned Seq Scan on public.t1
           Output: c1, c2
           Filter: ((t1.c1)::text = '12'::text)
           Selected Partitions:  1..3

(15 rows)
```

  - STABLE/VOLATILE functions in non-multi-primary mode:

```
gaussdb=# CREATE SEQUENCE seq;
gaussdb=# PREPARE p6(TEXT) AS SELECT * FROM t1 WHERE c1 = currval($1);-- This VOLATILE
function does not support pruning.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
PREPARE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) EXECUTE p6('seq');
                              QUERY PLAN
-----------------------------------------------------------------------------
 Data Node Scan
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE true
   Coordinator quals: ((t1.c1)::numeric = currval(('seq'::text)::regclass))

 Remote SQL: SELECT c1, c2 FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   Partition Iterator
     Output: c1, c2
     Iterations: 3
     ->  Partitioned Seq Scan on public.t1
           Output: c1, c2
           Selected Partitions:  1..3

(15 rows)

-- Clean up the environment.
gaussdb=# DROP TABLE t1;
```

Dynamic Pruning of Parameterized Paths

Dynamic pruning of parameterized paths is triggered when a parameterized path is generated on the partition-key column of a partitioned table (that column is joined to a column from another table, and the partitioned table serves as the inner table in a nested-loop index scan). This pruning takes place once each time the outer table scan is completed, just before the inner table scan is initiated.

NOTICE

- Dynamic pruning of parameterized paths does not support the following: subquery expressions (except when the pruning operator selects a parameterized path and the partition key contains an InitPlan subquery condition), STABLE or VOLATILE functions, cross–query block parameterized paths, or BitmapOr and BitmapAnd operators.
- When you query plans via EXPLAIN, the partition count shown for a parameterized path pruning plan represents the maximum possible number of partitions that could be accessed. The actual number of accessed partitions may be significantly smaller.
- When you query plans via EXPLAIN ANALYZE, the parameterized path pruning plans for partitioned tables will show "ppi-pruning, actual scanned:*XXX*". **ppi-pruning** indicates that parameterized path pruning has been triggered. **actual scanned** does not represent the parameterized path pruning result: In non-multi-primary mode, pruning results may vary across nodes; therefore, **actual scanned** only reflects the result after static and dynamic pruning.

- Typical scenarios that support dynamic parameterized path pruning:
  - Comparison expressions in non-multi-primary mode

```
-- Create partitioned tables and indexes.
gaussdb=# CREATE TABLE t1 (c1 INT, c2 INT)
PARTITION BY RANGE (c1)
(
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(20),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PARTITION p3 VALUES LESS THAN(MAXVALUE)
);
gaussdb=# CREATE TABLE t2 (c1 INT, c2 INT)
PARTITION BY RANGE (c1)
(
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(20),
    PARTITION p3 VALUES LESS THAN(MAXVALUE)
);
gaussdb=# CREATE INDEX t1_c1 ON t1(c1) LOCAL;
gaussdb=# CREATE INDEX t2_c1 ON t2(c1) LOCAL;
gaussdb=# CREATE INDEX t1_c2 ON t1(c2) LOCAL;
gaussdb=# CREATE INDEX t2_c2 ON t2(c2) LOCAL;

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT /*+ nestloop(t1 t2) indexscan(t1)
indexscan(t2) */ * FROM t2 JOIN t1 ON t1.c1 = t2.c1;
                                                 QUERY
PLAN
-------------------------------------------------------------------------------------------------------------------
-----------------------------------
 Data Node Scan
   Output: t2.c1, t2.c2, t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT/*+ NestLoop(t1 t2) IndexScan(t1) IndexScan(t2)*/ t2.c1, t2.c2, t1.c1,
t1.c2 FROM public.t2 JOIN public.t1 ON t1.c1 = t2.c1

 Remote SQL: SELECT/*+ NestLoop(t1 t2) IndexScan(t1) IndexScan(t2)*/ t2.c1, t2.c2, t1.c1, t1.c2
FROM public.t2 JOIN public.t1 ON t1.c1 = t2.c1
 Datanode Name: datanode1
   Nested Loop
     Output: t2.c1, t2.c2, t1.c1, t1.c2
     ->  Partition Iterator
           Output: t2.c1, t2.c2
           Iterations: 3
           ->  Partitioned Index Scan using t2_c1 on public.t2
                 Output: t2.c1, t2.c2
                 Selected Partitions:  1..3
     ->  Partition Iterator
           Output: t1.c1, t1.c2
           Iterations: PART
           ->  Partitioned Index Scan using t1_c1 on public.t1
                 Output: t1.c1, t1.c2
                 Index Cond: (t1.c1 = t2.c1)
                 Selected Partitions:  1..3 (ppi-pruning)

(23 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT /*+ nestloop(t1 t2) indexscan(t1)
indexscan(t2) */ * FROM t2 JOIN t1 ON t1.c1 < t2.c1;
                          QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t2.c1, t2.c2, t1.c1, t1.c2
   Node/s: All datanodes
   ->  Nested Loop
         Output: t2.c1, t2.c2, t1.c1, t1.c2
         ->  Streaming(type: BROADCAST)
               Output: t2.c1, t2.c2
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Partition Iterator
                     Output: t2.c1, t2.c2
                     Iterations: 3
                     ->  Partitioned Seq Scan on public.t2
                           Output: t2.c1, t2.c2
                           Distribute Key: t2.c1
                           Selected Partitions:  1..3
         ->  Partition Iterator
               Output: t1.c1, t1.c2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            Iterations: PART
            ->  Partitioned Index Scan using t1_c1 on public.t1
                  Output: t1.c1, t1.c2
                  Distribute Key: t1.c1
                  Index Cond: (t1.c1 < t2.c1)
                  Selected Partitions:  1..3 (ppi-pruning)
(24 rows)

gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT /*+ nestloop(t1 t2) indexscan(t1)
indexscan(t2) */ * FROM t2 JOIN t1 ON t1.c1 < t2.c1;
                        QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t2.c1, t2.c2, t1.c1, t1.c2
   Node/s: All datanodes
   ->  Nested Loop
         Output: t2.c1, t2.c2, t1.c1, t1.c2
         ->  Streaming(type: BROADCAST)
               Output: t2.c1, t2.c2
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Partition Iterator
                     Output: t2.c1, t2.c2
                     Iterations: 3
                     ->  Partitioned Seq Scan on public.t2
                           Output: t2.c1, t2.c2
                           Distribute Key: t2.c1
                           Selected Partitions:  1..3
         ->  Partition Iterator
               Output: t1.c1, t1.c2
               Iterations: PART
               ->  Partitioned Index Scan using t1_c1 on public.t1
                     Output: t1.c1, t1.c2
                     Distribute Key: t1.c1
                     Index Cond: (t1.c1 > t2.c1)
                     Selected Partitions:  1..3 (ppi-pruning)
(24 rows)
```

- Logical expressions in non-multi-primary mode

```
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT /*+ nestloop(t1 t2) indexscan(t1)
indexscan(t2) */ * FROM t2 JOIN t1 ON t1.c1 = t2.c1 AND t1.c2 = 2;
                                                    QUERY
PLAN
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
 Data Node Scan
   Output: t2.c1, t2.c2, t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT/*+ NestLoop(t1 t2) IndexScan(t1) IndexScan(t2)*/ t2.c1, t2.c2, t1.c1,
t1.c2 FROM public.t2 JOIN public.t1 ON t1.c1 = t2.c1 AND t1.c2 = 2

 Remote SQL: SELECT/*+ NestLoop(t1 t2) IndexScan(t1) IndexScan(t2)*/ t2.c1, t2.c2, t1.c1, t1.c2
FROM public.t2 JOIN public.t1 ON t1.c1 = t2.c1 AND t1.c2 = 2
 Datanode Name: datanode1
   Nested Loop
     Output: t2.c1, t2.c2, t1.c1, t1.c2
     ->  Partition Iterator
           Output: t1.c1, t1.c2
           Iterations: 3
           ->  Partitioned Index Scan using t1_c2 on public.t1
                 Output: t1.c1, t1.c2
                 Index Cond: (t1.c2 = 2)
                 Selected Partitions:  1..3
     ->  Partition Iterator
           Output: t2.c1, t2.c2
           Iterations: PART
           ->  Partitioned Index Scan using t2_c1 on public.t2
                 Output: t2.c1, t2.c2
                 Index Cond: (t2.c1 = t1.c1)
                 Selected Partitions:  1..3 (ppi-pruning)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(24 rows)
```

- Typical scenarios that do not support dynamic parameterized path pruning:
  - BitmapOr and BitmapAnd operators

```
gaussdb=# set enable_seqscan=off;
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT /*+ nestloop(t1 t2) */ * FROM t2 JOIN t1 ON t1.c1 = t2.c1 OR t1.c2 = 2;
WARNING:  Statistics in some tables or columns(public.t2.c1, public.t1.c1, public.t1.c2) are not collected.
HINT:  Do analyze for them in order to generate optimized plan.
                              QUERY PLAN
-----------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t2.c1, t2.c2, t1.c1, t1.c2
   Node/s: All datanodes
   ->  Nested Loop
         Output: t2.c1, t2.c2, t1.c1, t1.c2
         ->  Streaming(type: BROADCAST)
               Output: t2.c1, t2.c2
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Partition Iterator
                     Output: t2.c1, t2.c2
                     Iterations: 3
                     ->  Partitioned Seq Scan on public.t2
                           Output: t2.c1, t2.c2
                           Distribute Key: t2.c1
                           Selected Partitions:  1..3
         ->  Partition Iterator
               Output: t1.c1, t1.c2
               Iterations: 3
               ->  Partitioned Bitmap Heap Scan on public.t1
                     Output: t1.c1, t1.c2
                     Distribute Key: t1.c1
                     Recheck Cond: ((t1.c1 = t2.c1) OR (t1.c2 = 2))
                     Selected Partitions:  1..3
                     ->  BitmapOr
                           ->  Partitioned Bitmap Index Scan on t1_c1
                                 Index Cond: (t1.c1 = t2.c1)
                           ->  Partitioned Bitmap Index Scan on t1_c2
                                 Index Cond: (t1.c2 = 2)
(29 rows)
```

  - Implicit conversions in non-multi-primary mode

```
gaussdb=# CREATE TABLE t3(c1 TEXT, c2 INT);
CREATE TABLE
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 JOIN t3 ON t1.c1 = t3.c1;
WARNING:  Statistics in some tables or columns(public.t1.c1, public.t3.c1) are not collected.
HINT:  Do analyze for them in order to generate optimized plan.
                          QUERY PLAN
---------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.c1, t1.c2, t3.c1, t3.c2
   Node/s: All datanodes
   ->  Nested Loop
         Output: t1.c1, t1.c2, t3.c1, t3.c2
         Join Filter: (t1.c1 = (lengthb(t3.c1)))
         ->  Partition Iterator
               Output: t1.c1, t1.c2
               Iterations: 3
               ->  Partitioned Index Scan using t1_c1 on public.t1
                     Output: t1.c1, t1.c2
                     Distribute Key: t1.c1
                     Selected Partitions:  1..3
         ->  Materialize
               Output: t3.c1, t3.c2, (lengthb(t3.c1))
               ->  Streaming(type: REDISTRIBUTE)
                     Output: t3.c1, t3.c2, (lengthb(t3.c1))
                     Distribute Key: (lengthb(t3.c1))
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                Spawn on: All datanodes
                Consumer Nodes: All datanodes
                ->  Seq Scan on public.t3
                      Output: t3.c1, t3.c2, lengthb(t3.c1)
                      Distribute Key: t3.c1
(23 rows)
```

- Functions in non-multi-primary mode

```
gaussdb=# EXPLAIN (VERBOSE ON, COSTS OFF) SELECT * FROM t1 JOIN t3 ON t1.c1 =
LENGTHB(t3.c1);
                    QUERY PLAN
------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.c1, t1.c2, t3.c1, t3.c2
   Node/s: All datanodes
   ->  Hash Join
         Output: t1.c1, t1.c2, t3.c1, t3.c2
         Hash Cond: ((lengthb(t3.c1)) = t1.c1)
         ->  Streaming(type: REDISTRIBUTE)
               Output: t3.c1, t3.c2, (lengthb(t3.c1))
               Distribute Key: (lengthb(t3.c1))
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Seq Scan on public.t3
                     Output: t3.c1, t3.c2, lengthb(t3.c1)
                     Distribute Key: t3.c1
         ->  Hash
               Output: t1.c1, t1.c2
               ->  Partition Iterator
                     Output: t1.c1, t1.c2
                     Iterations: 3
                     ->  Partitioned Seq Scan on public.t1
                           Output: t1.c1, t1.c2
                           Distribute Key: t1.c1
                           Selected Partitions:  1..3
(23 rows)

-- Drop tables.
gaussdb=# DROP TABLE t1;
gaussdb=# DROP TABLE t2;
gaussdb=# DROP TABLE t3;
```

7.8.3.2 Optimized Execution of Partition Operators

7.8.3.2.1 Partition Iterator Elimination

Scenario

In the current partitioned table architecture, the executor iteratively accesses each partition through the Partition Iterator operator. However, when partition pruning results in a single partition, this operator becomes redundant as an iterator. In such cases, eliminating the operator can reduce unnecessary execution overhead. Given the executor's PIPELINE architecture, where the Partition Iterator operator is executed repeatedly, the performance gains from its elimination become particularly significant in data-intensive scenarios.

Examples

In non-multi-primary mode, activate partition iterator elimination by setting the GUC parameter **partition_iterator_elimination**. For example:

```
gaussdb=# CREATE TABLE t1 (c1 INT, c2 INT) PARTITION BY HASH(c1) PARTITIONS 2 SUBPARTITION BY
HASH (c2) SUBPARTITIONS 2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# INSERT INTO t1 SELECT generate_series(1,10),generate_series(1,10);
gaussdb=# SET max_datanode_for_plan = 1;

gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 = 7 AND c2 = 7;
                          QUERY PLAN
------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: datanode3

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = 7 AND c2 = 7
 Datanode Name: datanode3
   Partitioned Seq Scan on t1  (cost=0.00..42.23 rows=1 width=8)
     Filter: ((c1 = 7) AND (c2 = 7))
     Selected Partitions:  1
     Selected Subpartitions:  1:1

(10 rows)

gaussdb=# SET partition_iterator_elimination = off;
SET
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 = 7 AND c2 = 7;
                           QUERY PLAN
------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: datanode3

 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE c1 = 7 AND c2 = 7
 Datanode Name: datanode3
   Partition Iterator  (cost=0.00..42.23 rows=1 width=8)
     Iterations: 1, Sub Iterations: 1
     ->  Partitioned Seq Scan on t1  (cost=0.00..42.23 rows=1 width=8)
           Filter: ((c1 = 7) AND (c2 = 7))
           Selected Partitions:  1
           Selected Subpartitions:  1:1

(12 rows)

gaussdb=# DROP TABLE t1;
```

Precautions and Limitations

- This optimization takes effect in target scenarios only when the GUC parameter **partition_iterator_elimination** is enabled and the optimizer's pruning result yields a single partition.
- Cplan and certain Gplan scenarios are supported, such as when the partition key is defined as **a = $1** (a condition that permits single-partition pruning during the optimizer phase).
- The SeqScan, Indexscan, Indexonlyscan, Bitmapscan, RowToVec, and Tidscan operators are supported.
- In non-multi-primary mode, row store, Astore/Ustore, and SQL Bypass are supported.

7.8.3.2.2 Merge Append

Scenario

During global sorting of a partitioned table, the SQL engine typically performs a full scan in "Partition Iterator + PartitionScan" mode prior to sorting. This method makes it difficult to accelerate sorting by leveraging the local order within partitions. To solve this problem, partitioned tables introduce an improved sorting mechanism—Merge Append.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

The following is an example of executing Merge Append on a partitioned table in non-multi-primary mode:

```
gaussdb=# CREATE TABLE test_range_pt (a INT, b INT, c INT)
PARTITION BY RANGE(a)
(
   PARTITION p1 VALUES LESS THAN (2000),
   PARTITION p2 VALUES LESS THAN (3000),
   PARTITION p3 VALUES LESS THAN (4000),
   PARTITION p4 VALUES LESS THAN (5000),
   PARTITION p5 VALUES LESS THAN (MAXVALUE)
)ENABLE ROW MOVEMENT;
gaussdb=# INSERT INTO test_range_pt VALUES
(generate_series(1,10000),generate_series(1,10000),generate_series(1,10000));
gaussdb=# CREATE INDEX idx_range_b ON test_range_pt(b) LOCAL;
gaussdb=# ANALYZE test_range_pt;
gaussdb=# SET enable_stream_operator=off;
gaussdb=# SET max_datanode_for_plan=1;

gaussdb=# EXPLAIN SELECT * FROM test_range_pt  WHERE b >10 AND b < 5000 ORDER BY b LIMIT 10;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Limit  (cost=107.85..107.88 rows=10 width=12)
   ->  Sort  (cost=107.85..120.33 rows=10 width=12)
         Sort Key: test_range_pt.b
         ->  Data Node Scan on "__REMOTE_LIMIT_QUERY__"  (cost=0.00..0.00 rows=10 width=12)
               Node/s: All datanodes

 Remote SQL: SELECT a, b, c FROM ONLY public.test_range_pt WHERE b > 10 AND b < 5000 ORDER BY 2
LIMIT 10::bigint
 Datanode Name: datanode1
   Limit  (cost=0.06..1.15 rows=10 width=12)
     ->  Result  (cost=0.06..183.67 rows=10 width=12)
           ->  Merge Append  (cost=0.06..183.67 rows=10 width=12)
                 Sort Key: b
                 ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..19.36 rows=336
width=12)
                       Index Cond: ((b > 10) AND (b < 5000))
                       Selected Partitions:  1
                 ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..19.36 rows=336
width=12)
                       Index Cond: ((b > 10) AND (b < 5000))
                       Selected Partitions:  2
                 ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..19.36 rows=336
width=12)
                       Index Cond: ((b > 10) AND (b < 5000))
                       Selected Partitions:  3
                 ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..19.36 rows=336
width=12)
                       Index Cond: ((b > 10) AND (b < 5000))
                       Selected Partitions:  4
                 ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..19.36 rows=336
width=12)
                       Index Cond: ((b > 10) AND (b < 5000))
                       Selected Partitions:  5

(28 rows)

-- Disable the Merge Append operator for the partitioned table.
gaussdb=# SET sql_beta_feature = 'disable_merge_append_partition';
SET

gaussdb=# EXPLAIN SELECT * FROM test_range_pt WHERE b >10 AND b < 5000 ORDER BY b LIMIT 10;
                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Limit  (cost=107.85..107.88 rows=10 width=12)
   ->  Sort  (cost=107.85..120.33 rows=10 width=12)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
       Sort Key: test_range_pt.b
       ->  Data Node Scan on "__REMOTE_LIMIT_QUERY__"  (cost=0.00..0.00 rows=10 width=12)
            Node/s: All datanodes

 Remote SQL: SELECT a, b, c FROM ONLY public.test_range_pt WHERE b > 10 AND b < 5000 ORDER BY 2
LIMIT 10::bigint
 Datanode Name: datanode1
  Limit  (cost=102.89..102.91 rows=10 width=12)
    ->  Sort  (cost=102.89..107.08 rows=10 width=12)
          Sort Key: b
          ->  Partition Iterator  (cost=0.00..66.62 rows=1678 width=12)
                Iterations: 5
                ->  Partitioned Seq Scan on test_range_pt  (cost=0.00..66.62 rows=1678 width=12)
                      Filter: ((b > 10) AND (b < 5000))
                      Selected Partitions:  1..5

(17 rows)

gaussdb=# DROP TABLE test_range_pt;
```

Merge Append runs at a much lower cost than the common mode.

Precautions and Limitations

- Merge Append can be executed only during index scans or index-only scans.
- Merge Append can be executed only when the partition pruning result is greater than 1.
- Merge Append can be executed only when all partitioned indexes are valid B-tree indexes.
- Merge Append can be executed only when SQL statements contain the LIMIT clause.
- Merge Append cannot be executed when a filter is used for partition scans.
- The Merge Append path is no longer generated when the GUC parameter **sql_beta_feature** is set to **'disable_merge_append_partition'**.

7.8.3.2.3 Max/Min

Scenario

For partitioned tables using the Max or Min function, the SQL engine typically performs a full scan in "Partition Iterator + PartitionScan" mode, followed by a Sort + Limit operation. During an index scan for partitions, you can perform the Limit operation on each partition to calculate its Max or Min value, and then perform the Sort + Limit operation on the partitioned table. This approach reduces the data volume of the Sort operation to the number of partitions, since you already have the Max or Min value of each partition. As a result, the Sort overhead decreases significantly.

Examples

The following is an example of executing the Max/Min function on a partitioned table:

```
gaussdb=# CREATE TABLE test_range_pt (a INT, b INT, c INT)
PARTITION BY RANGE(a)
(
   PARTITION p1 VALUES LESS THAN (2000),
   PARTITION p2 VALUES LESS THAN (3000),
   PARTITION p3 VALUES LESS THAN (4000),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   PARTITION p4 VALUES LESS THAN (5000),
   PARTITION p5 VALUES LESS THAN (MAXVALUE)
)ENABLE ROW MOVEMENT;
gaussdb=# INSERT INTO test_range_pt VALUES(generate_series(1,10000), generate_series(1,10000),
generate_series(1,10000));
```

Before optimization in non-multi-primary mode:

```
gaussdb=# EXPLAIN ANALYZE SELECT min(b) FROM test_range_pt;
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=4.56..4.63 rows=1 width=8) (actual time=8.654..8.654 rows=1 loops=1)
   ->  Streaming (type: GATHER)  (cost=4.56..4.63 rows=3 width=8) (actual time=7.825..8.589 rows=3
loops=1)
         Node/s: All datanodes
         ->  Aggregate  (cost=4.50..4.50 rows=3 width=8) (actual time=[4.719,4.719]..[5.108,5.108], rows=3)
               ->  Partition Iterator  (cost=0.00..4.46 rows=30 width=4) (actual time=[0.110,3.964]..[0.114,4.356],
rows=10000)
                     Iterations: 5
                     ->  Partitioned Index Only Scan using idx_range_b on test_range_pt  (cost=0.00..4.46 rows=30
width=4) (actual time=[0.297,3.370]..[0.569,3.752], rows=10000)
                           Heap Fetches: 0
                           Selected Partitions:  1..5
 Total runtime: 10.287 ms
(10 rows)
```

After optimization in non-multi-primary mode:

```
gaussdb=# CREATE INDEX idx_range_b ON test_range_pt(b) LOCAL;
gaussdb=# EXPLAIN ANALYZE SELECT min(b) FROM test_range_pt;
                                                                       QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
 Result  (cost=0.64..0.65 rows=1 width=0) (actual time=2.806..2.806 rows=1 loops=1)
   InitPlan 1 (returns $2)
     ->  Aggregate  (cost=0.63..0.64 rows=1 width=4) (actual time=2.800..2.801 rows=1 loops=1)
           ->  Streaming (type: GATHER)  (cost=0.56..0.63 rows=3 width=4) (actual time=2.585..2.786 rows=3
loops=1)
                 Node/s: All datanodes
                 ->  Limit  (cost=0.50..0.50 rows=3 width=4) (actual time=[0.323,0.323]..[0.335,0.335], rows=3)
                       ->  Sort  (cost=0.50..0.52 rows=3 width=4) (actual time=[0.322,0.322]..[0.334,0.334], rows=3)
                             Sort Key: public.test_range_pt.b
                             Sort Method: top-N heapsort  Memory: 25kB ~ 25kB
                             ->  Partition Iterator  (cost=0.00..0.46 rows=15 width=4) (actual time=[0.081,0.294]..
[0.084,0.304], rows=15)
                                   Iterations: 5
                                   ->  Limit  (cost=0.00..0.46 rows=3 width=4) (actual time=[0.252,0.253]..
[0.257,0.257], rows=15)
                                         ->  Partitioned Index Only Scan using idx_range_b on test_range_pt
(cost=0.00..4.50 rows=3 width=4) (actual time=[0.252,0.252]..[0.255,0.255], rows=15)
                                               Index Cond: (b IS NOT NULL)
                                               Heap Fetches: 0
                                               Selected Partitions:  1..5
 Total runtime: 4.478 ms
(17 rows)
```

The time consumed after optimization is much shorter than that before optimization.

```
-- Drop the table.
gaussdb=# DROP TABLE test_range_pt;
```

The time consumed after optimization is much shorter than that before optimization.

```
-- Clean up.
gaussdb=# DROP TABLE test_range_pt;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions and Limitations

- Max/Min is supported only during index scans or index-only scans.
- Max/Min is supported only when all partitioned indexes are valid B-tree indexes.

7.8.3.2.4 Dynamic Pruning

Scenario

If the partition key contains an equality query condition, the scan range can be pruned to a single partition. Consequently, the database engine directly maps the partition route to the target partition to obtain partition information, implementing precise partition locating, improving dynamic pruning performance, and reducing the noise floor during scan operator initialization.

NOTE

Dynamic pruning performance optimization is not reflected in plans.

Examples in Non-Multi-Primary Mode

```
gaussdb=# CREATE TABLE t_list
(
    c1 INT,
    c2 INT
)
PARTITION BY LIST (c1)
(
    PARTITION p0 VALUES (1,2,3),
    PARTITION p1 VALUES (4,5,6),
    PARTITION p2 VALUES (7,8,9),
    PARTITION p3 VALUES (DEFAULT)
);
-- The Gplan is pruned to a single partition to optimize dynamic pruning performance.
gaussdb=# SET plan_cache_mode = 'force_generic_plan';
gaussdb=# PREPARE p1 AS SELECT /*+tablescan(t_list)*/ c2 FROM t_list WHERE c1=$1;
gaussdb=# EXPLAIN (verbose off, costs off) EXECUTE p1(1);
                              QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan
   Node/s: datanode1
   Node expr: $1

 Remote SQL: SELECT/*+ TableScan(t_list)*/ c2 FROM public.t_list WHERE c1 = $1
 Datanode Name: datanode1
   Partitioned Seq Scan on t_list
     Filter: (c1 = $1)
     Selected Partitions:  1

(10 rows)
gaussdb=# DROP TABLE t_list;
```

Precautions and Limitations

- This feature only applies to the scenario where the partition key contains an equality query condition and the scan range is pruned to a single partition.
- Dynamic pruning is limited to Gplans; Cplans do not support it.
- The partition key is limited to a single column; multi-column partition keys are not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The SeqScan, IndexScan, IndexonlyScan, and BitmapScan operators are supported.
- The SQLBypass scenario is not supported.
- Expression partition keys are not supported.

7.8.3.3 Partitioned Index

There are three types of indexes on a partitioned table:

- Global non-partitioned index (see **Figure 7-8**).
- Global partitioned index (see **Figure 7-9**).
- Local partitioned index (see **Figure 7-10**).

Currently, GaussDB supports global non-partitioned indexes and local partitioned indexes.

Figure 7-8 Global non-partitioned index

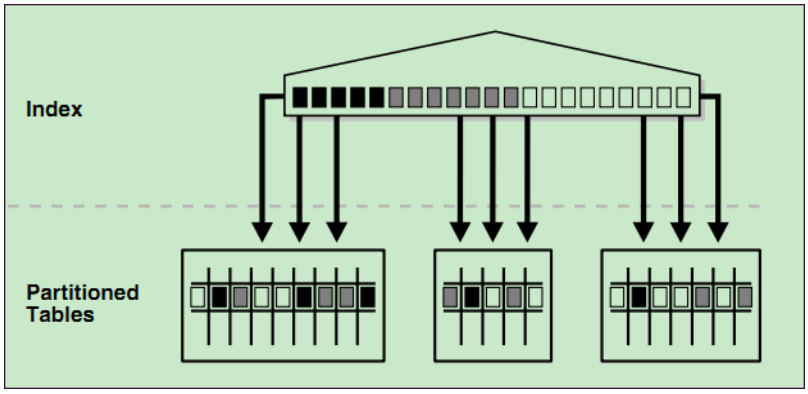
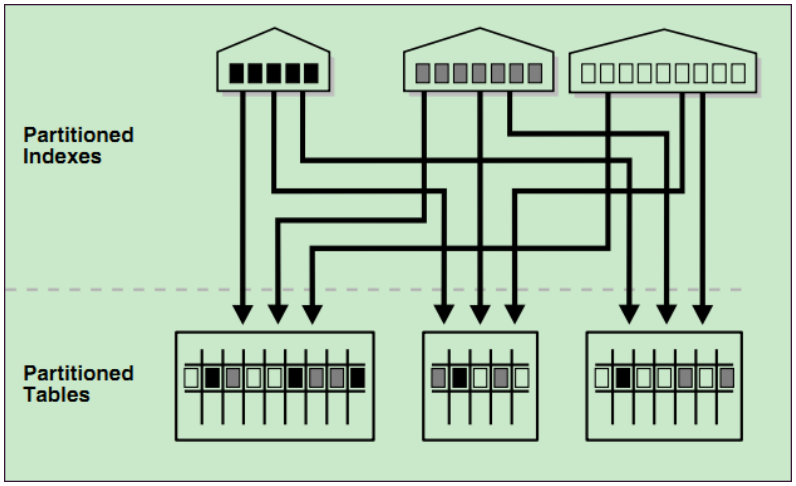
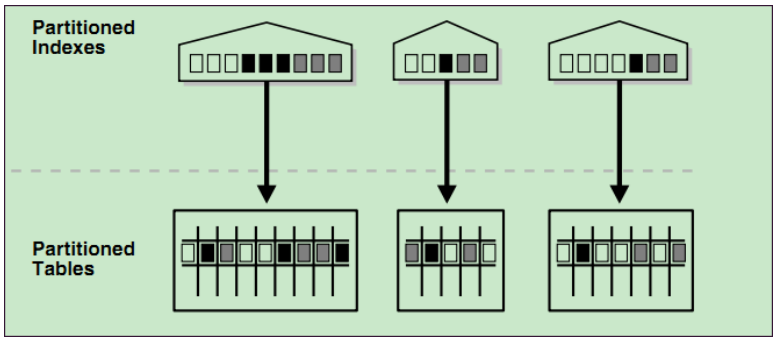


Figure 7-9 Global partitioned index



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-10 Local partitioned index



Limitations

- Indexes on partitioned tables are classified into local or global indexes. A local index is specific to a partition within a partitioned table, while a global index spans the entire partitioned table.
- When creating an index, you need to determine the index type based on the settings of UNIQUE and PRIMARY KEY constraints. If the constraint keys cover all partition keys, create a local index. Otherwise, create a global index.
- When creating a local index, you can use the FOR { partition_name | (partition_value [, ...]) } clause to specify a single partition. This type of index is limited to the specified partition—it does not take effect on other partitions and will not be automatically created on subsequent new partitions.
- Currently, the query paths of sparsely partitioned indexes can be generated only in the plan where the scan range is statically pruned to a single partition.

NOTE

If a query statement involves multiple target partitions, you are advised to use a global index. Otherwise, use a local index. Keep in mind that global indexes incur additional overhead for partition maintenance.

Examples

- Create a table.

```
gaussdb=# CREATE TABLE web_returns_p2
(
    ca_address_sk INTEGER NOT NULL ,
    ca_address_id CHARACTER(16) NOT NULL ,
    ca_street_number CHARACTER(10) ,
    ca_street_name CHARACTER VARYING(60) ,
    ca_street_type CHARACTER(15) ,
    ca_suite_number CHARACTER(10) ,
    ca_city CHARACTER VARYING(60) ,
    ca_county CHARACTER VARYING(30) ,
    ca_state CHARACTER(2) ,
    ca_zip CHARACTER(10) ,
    ca_country CHARACTER VARYING(20) ,
    ca_gmt_offset NUMERIC(5,2) ,
    ca_location_type CHARACTER(20)
)
PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

- Create an index.
  - Create the local index **tpcds_web_returns_p2_index1** without specifying a partition name.

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

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

    ```
    CREATE INDEX
    ```

  - Create the local index **tpcds_web_returns_p2_index2** with a specified partition name.

    ```
    gaussdb=# CREATE TABLESPACE example2 LOCATION '/home/omm/example2';
    gaussdb=# CREATE TABLESPACE example3 LOCATION '/home/omm/example3';
    gaussdb=# CREATE TABLESPACE example4 LOCATION '/home/omm/example4';

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

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

    ```
    CREATE INDEX
    ```

  - Create the global index **tpcds_web_returns_p2_global_index** for a partitioned table.

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_global_index ON web_returns_p2
    (ca_street_number) GLOBAL;
    ```

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

    ```
    CREATE INDEX
    ```

  - Create a sparsely partitioned index.

    Specify a partition name.

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_for_p1 ON web_returns_p2 (ca_address_id)
    LOCAL(partition ind_part for p1);
    ```

    Specify the value of the partition key.

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_for_p2 ON web_returns_p2 (ca_address_id)
    LOCAL(partition ind_part for (5000));
    ```

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

    ```
    CREATE INDEX
    ```

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

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

```
ALTER INDEX
```

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

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

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

```
ALTER INDEX
```

- Rename an index partition.

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

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

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

```
ALTER INDEX
```

- Query indexes.
  – Query all indexes defined by the system and users.

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

  – Query information about a specified index.

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

- Drop an index.

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

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

```
DROP INDEX
```

- Drop a table.

```
gaussdb=# DROP TABLE web_returns_p2;
```

7.8.3.4 Partitioned Table Statistics

For partitioned tables, you can collect partition-level statistics and query the statistics in the pg_partition and pg_statistic system catalogs, as well as the pg_stats and pg_ext_stats views.

Partition-level statistics apply to the scenario where static pruning narrows the scan range of a partitioned table to a single partition. The collected statistics include the number of pages and tuples, single-column statistics, multi-column statistics, and expression index statistics at the partition level.

You can collect statistics on partitioned tables in either of the following ways:

- Collecting statistics in cascading mode
- Collecting statistics on a specified partition

7.8.3.4.1 Collecting Statistics in Cascading Mode

Upon the execution of ANALYZE | ANALYSE on a partitioned table, the system automatically collects all partition-level statistics that meet semantic requirements, based on the specified or default **PARTITION_MODE**. For details, see the **PARTITION_MODE** parameter in "SQL Reference > SQL Syntax > A > ANALYZE | ANALYSE" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- In non-multi-primary mode, specifying **PARTITION_MODE=ALL** during cascading statistics collection for replication or hash bucket partitioned tables will automatically convert the setting to **ALL COMPLETE**.
- Cascading collection of partition-level statistics does not support the scenario where the value of **default_statistics_target** is a negative number.

Examples

- Create a partitioned table and insert data.

```
gaussdb=# CREATE TABLE t1_range_int
(
    c1 INT,
    c2 INT,
    c3 INT,
    c4 INT
)
PARTITION BY RANGE(c1)
(
    PARTITION range_p00 VALUES LESS THAN(10),
    PARTITION range_p01 VALUES LESS THAN(20),
    PARTITION range_p02 VALUES LESS THAN(30),
    PARTITION range_p03 VALUES LESS THAN(40),
    PARTITION range_p04 VALUES LESS THAN(50)
);
gaussdb=# INSERT INTO t1_range_int SELECT v,v,v,v FROM generate_series(0, 49) AS v;
```

- Collect statistics in cascading mode.

```
gaussdb=# ANALYZE t1_range_int WITH ALL;
```

- View partition-level statistics.

```
gaussdb=# SELECT relname, parttype, relpages, reltuples FROM pg_partition WHERE
parentid=(SELECT oid FROM pg_class WHERE relname='t1_range_int') ORDER BY relname;
   relname    | parttype | relpages | reltuples
--------------+----------+----------+-----------
 range_p00    | p        |        3 |        15
 range_p01    | p        |        3 |        12
 range_p02    | p        |        2 |        10
 range_p03    | p        |        0 |         0
 range_p04    | p        |        3 |        15
 t1_range_int | r        |        0 |         0
(6 rows)

gaussdb=# SELECT
schemaname,tablename,partitionname,subpartitionname,attname,inherited,null_frac,avg_width,n_disti
nct,n_dndistinct,most_common_vals,most_common_freqs,histogram_bounds FROM pg_stats WHERE
tablename='t1_range_int' ORDER BY tablename, partitionname, attname;
 schemaname |   tablename    | partitionname | subpartitionname | attname | inherited | null_frac |
avg_width | n_distinct | n_dndistinct | most_common_vals | most_common_freqs
|                                                histogram_bounds

------------+--------------+---------------+------------------+---------+-----------+-----------+-----------
+------------+--------------+------------------+-------------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------
 public     | t1_range_int | range_p00     |                  | c1      | f         |         0 |         4 |        -1 |
-1 |              |                  | {0,1,2,3,4,5,6,7,8,9}
 public     | t1_range_int | range_p00     |                  | c2      | f         |         0 |         4 |        -1 |
-1 |              |                  | {0,1,2,3,4,5,6,7,8,9}
 public     | t1_range_int | range_p00     |                  | c3      | f         |         0 |         4 |        -1 |
-1 |              |                  | {0,1,2,3,4,5,6,7,8,9}
 public     | t1_range_int | range_p00     |                  | c4      | f         |         0 |         4 |        -1 |
-1 |              |                  | {0,1,2,3,4,5,6,7,8,9}
 public     | t1_range_int | range_p01     |                  | c1      | f         |         0 |         4 |        -1 |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-1 |              |                | {10,11,12,13,14,15,16,17,18,19}
 public    | t1_range_int | range_p01    |              | c2     | f       |       0 |       4 |       -1 |
-1 |              |                | {10,11,12,13,14,15,16,17,18,19}
 public    | t1_range_int | range_p01    |              | c3     | f       |       0 |       4 |       -1 |
-1 |              |                | {10,11,12,13,14,15,16,17,18,19}
 public    | t1_range_int | range_p01    |              | c4     | f       |       0 |       4 |       -1 |
-1 |              |                | {10,11,12,13,14,15,16,17,18,19}
 public    | t1_range_int | range_p02    |              | c1     | f       |       0 |       4 |       -1 |
-1 |              |                | {20,21,22,23,24,25,26,27,28,29}
 public    | t1_range_int | range_p02    |              | c2     | f       |       0 |       4 |       -1 |
-1 |              |                | {20,21,22,23,24,25,26,27,28,29}
 public    | t1_range_int | range_p02    |              | c3     | f       |       0 |       4 |       -1 |
-1 |              |                | {20,21,22,23,24,25,26,27,28,29}
 public    | t1_range_int | range_p02    |              | c4     | f       |       0 |       4 |       -1 |
-1 |              |                | {20,21,22,23,24,25,26,27,28,29}
 public    | t1_range_int | range_p03    |              | c1     | f       |       0 |       4 |       -1 |
-1 |              |                | {30,31,32,33,34,35,36,37,38,39}
 public    | t1_range_int | range_p03    |              | c2     | f       |       0 |       4 |       -1 |
-1 |              |                | {30,31,32,33,34,35,36,37,38,39}
 public    | t1_range_int | range_p03    |              | c3     | f       |       0 |       4 |       -1 |
-1 |              |                | {30,31,32,33,34,35,36,37,38,39}
 public    | t1_range_int | range_p03    |              | c4     | f       |       0 |       4 |       -1 |
-1 |              |                | {30,31,32,33,34,35,36,37,38,39}
 public    | t1_range_int | range_p04    |              | c1     | f       |       0 |       4 |       -1 |
-1 |              |                | {40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int | range_p04    |              | c2     | f       |       0 |       4 |       -1 |
-1 |              |                | {40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int | range_p04    |              | c3     | f       |       0 |       4 |       -1 |
-1 |              |                | {40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int | range_p04    |              | c4     | f       |       0 |       4 |       -1 |
-1 |              |                | {40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int |              |              | c1     | f       |       0 |       4 |       -1 |        -1
|              |                |
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int |              |              | c2     | f       |       0 |       4 |       -1 |        -1
|              |                |
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int |              |              | c3     | f       |       0 |       4 |       -1 |        -1
|              |                |
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49}
 public    | t1_range_int |              |              | c4     | f       |       0 |       4 |       -1 |        -1
|              |                |
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49}
(24 rows)
```

- Generate partition-level statistics of data in multiple columns.

```
gaussdb=# ALTER TABLE t1_range_int ADD STATISTICS ((c2, c3));
gaussdb=# ANALYZE t1_range_int WITH ALL;
```

- View partition-level statistics of data in multiple columns.

```
gaussdb=# SELECT
schemaname,tablename,partitionname,subpartitionname,attname,inherited,null_frac,avg_width,n_disti
nct,n_dndistinct,most_common_vals,most_common_freqs,histogram_bounds FROM pg_ext_stats
WHERE tablename='t1_range_int' ORDER BY tablename,partitionname,attname;
 schemaname |   tablename    | partitionname | subpartitionname | attname | inherited | null_frac |
avg_width | n_distinct | n_dndistinct | most_common_vals | most_common_freqs | histogram_bounds
------------+--------------+---------------+------------------+---------+-----------+-----------+-----------
+------------+--------------+------------------+------------------+------------------
 public    | t1_range_int | range_p00    |              | 2 3    | f       |       0 |       8 |       -1 |
-1 |              |                |
 public    | t1_range_int | range_p01    |              | 2 3    | f       |       0 |       8 |       -1 |
-1 |              |                |
 public    | t1_range_int | range_p02    |              | 2 3    | f       |       0 |       8 |       -1 |
-1 |              |                |
 public    | t1_range_int | range_p03    |              | 2 3    | f       |       0 |       8 |       -1 |
-1 |              |                |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 public     | t1_range_int | range_p04     |                 | 2 3     | f         |         0 |         8 |         -1 |
-1 |                 |                 |
 public     | t1_range_int |               |                 | 2 3     | f         |         0 |         8 |         -1 |          -1
|                 |                 |
(6 rows)
```

- Create an expression index and generate relevant partition-level statistics.

```
gaussdb=# CREATE INDEX t1_range_int_index ON t1_range_int(text(c1)) LOCAL;
gaussdb=# ANALYZE t1_range_int WITH ALL;
```

- View the partition-level statistics of the expression index.

```
gaussdb=# SELECT
schemaname,tablename,partitionname,subpartitionname,attname,inherited,null_frac,avg_width,n_disti
nct,n_dndistinct,most_common_vals,most_common_freqs,histogram_bounds FROM pg_stats WHERE
tablename='t1_range_int_index' ORDER BY tablename,partitionname,attname;
 schemaname |      tablename       |    partitionname     | subpartitionname | attname | inherited |
null_frac | avg_width | n_distinct | n_dndistinct | most_common_vals | most_common_freqs
|                                                  histogram_bounds

------------+----------------------+----------------------+------------------+---------+-----------+-----------
+-----------+------------+--------------+------------------+------------------
+-----------------------------------------------------------------------------------
-------------------------------------------------------------
 public     | t1_range_int_index | range_p00_text_idx |                  | text    | f         |         0 |         5
|         -1 |                0 |                  |                  | {0,1,2,3,4,5,6,7,8,9}
 public     | t1_range_int_index | range_p01_text_idx |                  | text    | f         |         0 |         6
|         -1 |                0 |                  |                  | {10,11,12,13,14,15,16,17,18,19}
 public     | t1_range_int_index | range_p02_text_idx |                  | text    | f         |         0 |         6
|         -1 |                0 |                  |                  | {20,21,22,23,24,25,26,27,28,29}
 public     | t1_range_int_index | range_p03_text_idx |                  | text    | f         |         0 |         6
|         -1 |                0 |                  |                  | {30,31,32,33,34,35,36,37,38,39}
 public     | t1_range_int_index | range_p04_text_idx |                  | text    | f         |         0 |         6
|         -1 |                0 |                  |                  | {40,41,42,43,44,45,46,47,48,49}
 public     | t1_range_int_index |                    |                  | text    | f         |         0 |         5 |        -1
|          0 |                  |                  |
{0,1,10,11,12,13,14,15,16,17,18,19,2,20,21,22,23,24,25,26,27,28,29,3,30,31,32,33,3
4,35,36,37,38,39,4,40,41,42,43,44,45,46,47,48,49,5,6,7,8,9}
(6 rows)
```

- Drop the partitioned table.

```
gaussdb=# DROP TABLE t1_range_int;
```

7.8.3.4.2 Collecting Partition-Level Statistics

Collecting Statistics on a Specified Partition

You can collect statistics on a specified partition within a partitioned table. When you collect statistics again, the system automatically updates and maintains the statistics for that partition. This feature applies to list, hash, and range partitions.

```
gaussdb=# CREATE TABLE only_fisrt_part(id int,name varchar)PARTITION BY RANGE (id)
 (PARTITION id11 VALUES LESS THAN (1000000),
PARTITION id22 VALUES LESS THAN (2000000),
PARTITION max_id1 VALUES LESS THAN (MAXVALUE));

gaussdb=# INSERT INTO only_fisrt_part SELECT generate_series(1,5000),'test';

gaussdb=# ANALYZE only_fisrt_part PARTITION (id11);
gaussdb=# ANALYZE only_fisrt_part PARTITION (id22);
gaussdb=# ANALYZE only_fisrt_part PARTITION (max_id1);

gaussdb=# SELECT relname, relpages, reltuples FROM pg_partition WHERE relname IN ('id11', 'id22',
'max_id1');
 relname | relpages | reltuples
---------+----------+-----------
 id11    |     3400 |      5000
 id22    |        0 |         0
 max_id1 |        0 |         0
```

```
(3 rows)

gaussdb=# \x
gaussdb=# SELECT * FROM pg_stats WHERE tablename ='only_fisrt_part' AND partitionname ='id11';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
schemaname             | public
tablename              | only_fisrt_part
attname                | name
inherited              | f
null_frac              | 0
avg_width              | 5
n_distinct             | 1
n_dndistinct           | 1
most_common_vals       | {test}
most_common_freqs      | {1}
histogram_bounds       |
correlation            | 1
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
partitionname          | id11
subpartitionname       |
-[ RECORD 2 ]----------
+-----------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
schemaname             | public
tablename              | only_fisrt_part
attname                | id
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | -1
n_dndistinct           | 1
most_common_vals       |
most_common_freqs      |
histogram_bounds       |
{1,50,100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,1000,1050,1100,1150,1200,
1250,1300,1350,1400,1450,1500,1550,1600,1650,1700,1750,1800,1850,1900,1950,2000,2050,2100,2150,2200,
2250,2300,2350,2400,2450,2500,2550,2600,2650,2700,2750,2800,2850,2900,2950,3000,3050,3100,3150,3200,
3250,3300,3350,3400,3450,3500,3550,3600,3650,3700,3750,3800,3850,3900,3950,4000,4050,4100,4150,4200,
4250,4300,4350,4400,4450,4500,4550,4600,4650,4700,4750,4800,4850,4900,4950,5000}
correlation            | 1
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
partitionname          | id11
subpartitionname       |
gaussdb=# q \x
-- Drop the partitioned table.
gaussdb=# DROP TABLE only_fisrt_part;
```

Using Statistics on a Specified Partition for the Optimizer

When processing a partitioned table, the optimizer preferentially uses the statistics of the specified partition to optimize queries. However, if no statistics have been collected for the specified partition, the optimizer rewrites the partition clause to optimize pruning. For details, see **Rewriting a Partition Clause to Optimize Pruning**.

```
gaussdb=# SET enable_fast_query_shipping = off;
gaussdb=# CREATE TABLE ONLY_FIRST_PART_TWO
(
```

```
    C1 INT,
    C2 BIGINT
)
PARTITION BY RANGE(C1)
(
    PARTITION P_1 VALUES LESS THAN (1000),
    PARTITION P_2 VALUES LESS THAN (3000),
    PARTITION P_3 VALUES LESS THAN (MAXVALUE)
);

gaussdb=# INSERT INTO only_first_part_two SELECT generate_series(1,5000), 0;
gaussdb=# EXPLAIN SELECT * FROM only_first_part_two PARTITION (p_2);
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.81..5.35 rows=20 width=12)
   Node/s: All datanodes
   ->  Partitioned Seq Scan on only_first_part_two  (cost=0.00..4.13 rows=20 width=12)
         Selected Partitions:  2
(4 rows)

gaussdb=# EXPLAIN SELECT * FROM only_first_part_two PARTITION (p_1) where c2 = 2;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..4.26 rows=1 width=12)
   Node/s: All datanodes
   ->  Partitioned Seq Scan on only_first_part_two  (cost=0.00..4.16 rows=1 width=12)
         Filter: (c2 = 2)
         Selected Partitions:  1
(5 rows)

gaussdb=# DROP TABLE only_first_part_two;
```

Rewriting a Partition Clause to Optimize Pruning

If partition-level statistics are unavailable, the optimizer rewrites the pseudo-predicate of the partition clause logically and uses the rewritten pseudo-predicate to influence selectivity calculation and produce a more accurate row count estimate based on the statistics of the entire table.

NOTE

- This feature applies only to selectivity calculation.
- In non-multi-primary mode, this feature does not support subpartitions.
- In non-multi-primary mode, this feature supports only range and list partitions.
- For range partitions, this feature rewrites only single-column partition keys and does not support multi-column partition keys.
- For list partitions, this feature requires each specified partition to have no more than 40 enumerated values to maintain performance.
  - When a partition includes more than 40 enumerated values, this feature will not apply.
  - For the default partition, the number of enumerated values equals the total number of enumerated values in all non-default partitions.

Example: Rewriting a range partition in non-multi-primary mode

```
gaussdb=# CREATE TABLE test_int4_maxvalue(id INT, name VARCHAR)
PARTITION BY RANGE(id)
(
    PARTITION id1 VALUES LESS THAN(1000),
    PARTITION id2 VALUES LESS THAN(2000),
    PARTITION max_id VALUES LESS THAN(MAXVALUE)
);
gaussdb=# INSERT INTO test_int4_maxvalue SELECT GENERATE_SERIES(1,5000),'test';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# ANALYZE test_int4_maxvalue with global;

gaussdb=# set max_datanode_for_plan=2;
gaussdb=# set enable_stream_operator=off;

-- Query the specified partition id1.
gaussdb=# EXPLAIN SELECT * FROM test_int4_maxvalue PARTITION(id1);
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT id, name FROM ONLY public.test_int4_maxvalue PARTITION(id1) WHERE true
 Datanode Name: datanode1
   Partitioned Seq Scan on test_int4_maxvalue  (cost=0.00..28.61 rows=2561 width=9)
     Selected Partitions:  1

 Datanode Name: datanode2
   Partitioned Seq Scan on test_int4_maxvalue  (cost=0.00..27.39 rows=2439 width=9)
     Selected Partitions:  1

(12 rows)

-- Query the specified partition max_id.
gaussdb=# EXPLAIN SELECT * FROM test_int4_maxvalue PARTITION(max_id);
                                 QUERY PLAN
---------------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT id, name FROM ONLY public.test_int4_maxvalue PARTITION(max_id) WHERE true
 Datanode Name: datanode1
   Partitioned Seq Scan on test_int4_maxvalue  (cost=0.00..28.61 rows=2561 width=9)
     Selected Partitions:  3

 Datanode Name: datanode2
   Partitioned Seq Scan on test_int4_maxvalue  (cost=0.00..27.39 rows=2439 width=9)
     Selected Partitions:  3

(12 rows)

-- Drop the partitioned table.
gaussdb=# DROP TABLE test_int4_maxvalue;
```

Example: Rewriting a list partition in non-multi-primary mode

```
gaussdb=# CREATE TABLE test_default
(
    c1 INT,
    c2 BIGINT
)
PARTITION BY LIST(c2)
(
    PARTITION p_1 VALUES (10000, 20000),
    PARTITION p_2 VALUES (300000, 400000, 500000),
    PARTITION p_3 VALUES (DEFAULT)
);
gaussdb=# INSERT INTO test_default SELECT GENERATE_SERIES(1, 1000), 10000;
gaussdb=# INSERT INTO test_default SELECT GENERATE_SERIES(1001, 2000), 600000;
gaussdb=# ANALYZE test_default with global;

-- Query the specified partition p_1.
gaussdb=# EXPLAIN SELECT * FROM test_default PARTITION(p_1);
                               QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT c1, c2 FROM ONLY public.test_default PARTITION(p_1) WHERE true
 Datanode Name: datanode1
   Partitioned Seq Scan on test_default  (cost=0.00..11.16 rows=1016 width=12)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    Selected Partitions:  1

 Datanode Name: datanode2
   Partitioned Seq Scan on test_default  (cost=0.00..10.84 rows=984 width=12)
     Selected Partitions:  1

(12 rows)

-- Query the specified partition p_3.
gaussdb=# EXPLAIN SELECT * FROM test_default PARTITION(p_3);
                              QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT c1, c2 FROM ONLY public.test_default PARTITION(p_3) WHERE true
 Datanode Name: datanode1
   Partitioned Seq Scan on test_default  (cost=0.00..11.16 rows=1016 width=12)
     Selected Partitions:  3

 Datanode Name: datanode2
   Partitioned Seq Scan on test_default  (cost=0.00..10.84 rows=984 width=12)
     Selected Partitions:  3

(12 rows)

-- Drop the partitioned table.
gaussdb=# DROP TABLE test_default;
```

7.8.3.5 Partition-wise Join

Partition-wise join is a partition-level parallel optimization technique that breaks a join between two tables into smaller joins between corresponding partitions—one from each table—when specific conditions are met. This technique improves the performance of join queries on partitioned tables by enabling concurrent execution and downsizing data communication.

Partition-wise join is classified into symmetric multi-processing (SMP) and non-SMP scenarios.

7.8.3.5.1 Partition-wise Join in Non-SMP Scenarios

In non-SMP scenarios, the system generates a partition-wise join path based on predefined rules. In other words, it creates this join path whenever the conditions are met, without comparing path costs. To enable this feature, set the GUC parameter **enable_partitionwise**.

Specifications

Here are the partition-wise join specifications in non-SMP scenarios:

- Only range partitions are supported.
- Hash join, nested-loop join, and merge join are supported.
- Only inner join is supported.
- **query_dop** must be set to **1**.
- In non-SMP scenarios, partition-wise join selects paths based on predefined rules, which may degrade performance. You can determine whether to enable it based on actual needs.
- In non-multi-primary mode, only the FQS plan is supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

In non-multi-primary mode:

```
-- Create a range partitioned table.
gaussdb=# CREATE TABLE range_part (
    a INTEGER,
    b INTEGER,
    c INTEGER
) PARTITION BY RANGE (a)
(
    PARTITION range_part_p1 VALUES LESS THAN (10),
    PARTITION range_part_p2 VALUES LESS THAN (20),
    PARTITION range_part_p3 VALUES LESS THAN (30),
    PARTITION range_part_p4 VALUES LESS THAN (40)
);
CREATE TABLE

-- Enable the FQS plan.
gaussdb=# SET enable_fast_query_shipping= ON;
SET

-- Set query_dop to 1 and disable SMP.
gaussdb=# SET query_dop = 1;
SET

-- Disable partition-wise join in non-SMP scenarios.
gaussdb=# SET enable_partitionwise = off;
SET

-- View the non-partition-wise join plan.
gaussdb=# SET max_datanode_for_plan = 1;
SET
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM range_part t1 INNER JOIN range_part t2 ON (t1.a = t2.a);
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Node/s: All datanodes

 Remote SQL: SELECT t1.a, t1.b, t1.c, t2.a, t2.b, t2.c FROM public.range_part t1 JOIN public.range_part t2 ON
t1.a = t2.a
 Datanode Name: datanode1
   Hash Join
     Hash Cond: (t1.a = t2.a)
     ->  Partition Iterator
           Iterations: 4
           ->  Partitioned Seq Scan on range_part t1
                 Selected Partitions:  1..4
     ->  Hash
           ->  Partition Iterator
                 Iterations: 4
                 ->  Partitioned Seq Scan on range_part t2
                       Selected Partitions:  1..4

(17 rows)

-- Enable partition-wise join in non-SMP scenarios.
gaussdb=# SET enable_partitionwise = on;
SET

-- View the partition-wise join plan in non-SMP scenarios. The plan shows that the Partition Iterator
operator now sits above the Hash Join operator. Instead of scanning all partitions sequentially before
joining, the system now scans and joins one pair of partitions at a time and then moves on to the next pair.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM range_part t1 INNER JOIN range_part t2 ON (t1.a = t2.a);
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Remote SQL: SELECT t1.a, t1.b, t1.c, t2.a, t2.b, t2.c FROM public.range_part t1 JOIN public.range_part t2 ON
t1.a = t2.a
 Datanode Name: datanode1
   Result
     ->  Partition Iterator
           Iterations: 4
           ->  Hash Join
                 Hash Cond: (t1.a = t2.a)
                 ->  Partitioned Seq Scan on range_part t1
                       Selected Partitions:  1..4
                 ->  Hash
                       ->  Partitioned Seq Scan on range_part t2
                             Selected Partitions:  1..4

(16 rows)

-- Drop the partitioned table.
gaussdb=# DROP TABLE range_part;
```

7.8.3.5.2 Full Partition-wise Join in SMP Scenarios

In SMP scenarios, the partition-wise join plan is selected based on costs. During path generation, the system compares the estimated costs of partition-wise and non-partition-wise join paths and chooses the one with the lower cost. To enable this feature, set the GUC parameter **enable_smp_partitionwise**.

Partition-wise join in SMP scenarios is classified into full partition-wise join and partial partition-wise join.

- Full partition-wise join applies to two tables with the same partitioning policy.
- A full partition-wise join path is generated only when the partition keys of both tables are a pair of matching join keys.

Specifications

Here are the full partition-wise join specifications in SMP scenarios:

- Hash and range partitioned tables are supported.
- For hash partitioned tables, having the same partitioning policy means their partition key types and the number of partitions are identical.
- For range partitioned tables, having the same partitioning policy means their partition key types, number of partitions, number of partition keys, and boundary values of each partition are identical.
- In non-multi-primary mode, only the Stream plan is supported.
- In non-multi-primary mode, Join operators must execute entirely on a single DN. That is, their data cannot be distributed across multiple nodes.
- Hash join and nested-loop join are supported.
- SeqScan, IndexScan, IndexonlyScan and ImcvScan are supported. For IndexScan and IndexonlyScan, only partitioned local indexes are supported, and the index type must be B-tree or UB-tree.
- Related specifications are inherited from SMP scenarios. IUD operations are not supported in SMP scenarios.
- The SMP feature must be enabled, and **query_dop** must be greater than **1**.
- This feature retains the specifications present in the Stream plan and SMP scenarios.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Limitations

- Full partition-wise join cannot operate on intermediate result tables in range distribution scenarios.
- Partition-wise join is not supported when a partitioned table is pruned to a single partition.
- Partition-wise join executes solely within the same node group.
- Partition-wise join cannot operate on bucket tables or temporary tables.
- Partition-wise join is not supported when a DN is specified for a partition.
- Partition-wise join cannot operate on subpartitions.
- Only the Stream plan is supported. The light-proxy, FQS, and PGXC plans are not supported.
- Execution performance may degrade under data skew or partition pruning.
- Partition-wise join is prohibited in computational skew scenarios.

Examples

In non-multi-primary mode:

```
-- Create a hash partitioned table where the distribution key and partition key are the same.
gaussdb=# CREATE TABLE hash_part
(
    a INTEGER,
    b INTEGER,
    c INTEGER
)
DISTRIBUTE BY HASH(a)
PARTITION BY HASH(a)
(
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4,
    PARTITION p5
);
CREATE TABLE

-- Create a hash partitioned table where the distribution key and partition key are different.
gaussdb=# CREATE TABLE hash_part_diff_key
(
    a INTEGER,
    b INTEGER,
    c INTEGER
)
DISTRIBUTE BY HASH(a)
PARTITION BY HASH(b)
(
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4,
    PARTITION p5
);
CREATE TABLE

-- Create a range partitioned table where the distribution key and partition key are different.
gaussdb=# CREATE TABLE range_part_diff_key
(
    a INTEGER,
    b INTEGER,
    c INTEGER,
    d INTEGER,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    e INTEGER
) DISTRIBUTE BY RANGE(a) (
    SLICE p1 VALUES LESS THAN (100),
    SLICE p2 VALUES LESS THAN (200),
    SLICE p3 VALUES LESS THAN (300),
    SLICE p4 VALUES LESS THAN (400),
    SLICE p5 VALUES LESS THAN (500)
)
PARTITION BY RANGE(b) (
    PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200),
    PARTITION p3 VALUES LESS THAN (300),
    PARTITION p4 VALUES LESS THAN (400),
    PARTITION p5 VALUES LESS THAN (500)
);
CREATE TABLE

-- Enable the Stream plan.
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# SET enable_stream_operator = on;
SET

-- Set query_dop to 5 and enable SMP in debug mode.
gaussdb=# SET query_dop = 5;
SET

-- Disable the nested-loop plan.
gaussdb=# SET enable_material=off;
SET
gaussdb=# SET enable_broadcast=off;
SET
gaussdb=# SET enable_nestloop=off;
SET

-- Insert 100,000 basic test data records. (a, b, and c are random integers to ensure the randomness of hash distribution.)
gaussdb=# INSERT INTO hash_part (a, b, c)
SELECT
  FLOOR(RANDOM() * 1000000)::INTEGER AS a,
  FLOOR(RANDOM() * 500000)::INTEGER AS b,
  FLOOR(RANDOM() * 1000)::INTEGER AS c
FROM GENERATE_SERIES(1, 100000);
INSERT 0 100000

-- Repeat five times, doubling the data volume each time. The final total data volume reaches 3,200,000 (100,000 × 2^5).
gaussdb=# DO $$
BEGIN
  FOR i IN 1..5 LOOP
    INSERT INTO hash_part (a, b, c)
    SELECT a, b, c FROM hash_part;
  END LOOP;
END
$$;
ANONYMOUS BLOCK EXECUTE

gaussdb=# INSERT INTO hash_part_diff_key(a, b, c)
SELECT
  FLOOR(RANDOM() * 1000000)::INTEGER AS a,
  FLOOR(RANDOM() * 500000)::INTEGER AS b,
  FLOOR(RANDOM() * 1000)::INTEGER AS c
FROM GENERATE_SERIES(1, 100000);
INSERT 0 100000

gaussdb=# DO $$
BEGIN
  FOR i IN 1..5 LOOP
    INSERT INTO hash_part_diff_key (a, b, c)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    SELECT a, b, c FROM hash_part_diff_key;
  END LOOP;
END
$$;
ANONYMOUS BLOCK EXECUTE

gaussdb=# INSERT INTO range_part_diff_key(a, b, c)
SELECT
  FLOOR(RANDOM() * 500)::INTEGER AS a,
  FLOOR(RANDOM() * 500)::INTEGER AS b,
  FLOOR(RANDOM() * 500)::INTEGER AS c
FROM GENERATE_SERIES(1, 100000);
INSERT 0 100000

gaussdb=# DO $$
BEGIN
  FOR i IN 1..5 LOOP
    INSERT INTO range_part_diff_key(a, b, c)
    SELECT a, b, c FROM range_part_diff_key;
  END LOOP;
END
$$;
ANONYMOUS BLOCK EXECUTE

-- Analyze data.
gaussdb=# ANALYZE;
ANALYZE

-- Disable partition-wise join in SMP scenarios.
gaussdb=# SET enable_smp_partitionwise = off;
SET

-- View the non-partition-wise join plan. The plan shows that after two layers of operators (Partition Iterator and Partitioned Seq Scan) complete data scans, the Streaming(type: LOCAL REDISTRIBUTE) operator redistributes matched data to the same Join operator.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM hash_part t1, hash_part t2 WHERE t1.a = t2.a;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join
               Hash Cond: (t1.a = t2.a)
               ->  Streaming(type: LOCAL REDISTRIBUTE dop: 5/5)
                     Spawn on: All datanodes
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part t1
                                 Selected Partitions:  1..5
               ->  Hash
                     ->  Streaming(type: LOCAL REDISTRIBUTE dop: 5/5)
                           Spawn on: All datanodes
                           ->  Partition Iterator
                                 Iterations: 5
                                 ->  Partitioned Seq Scan on hash_part t2
                                       Selected Partitions:  1..5
(19 rows)

-- Enable partition-wise join in SMP scenarios.
gaussdb=# SET enable_smp_partitionwise = on;
SET


-- View the partition-wise join plan. The plan removes the Streaming operator, eliminating the need to redistribute data between threads. This reduces data transfer overhead and improves join performance.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM hash_part t1, hash_part t2 WHERE t1.a = t2.a;
                          QUERY PLAN
------------------------------------------------------------------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join (Partition-wise Join)
               Hash Cond: (t1.a = t2.a)
               ->  Partition Iterator
                     Iterations: 5
                     ->  Partitioned Seq Scan on hash_part t1
                           Selected Partitions:  1..5
               ->  Hash
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part t2
                                 Selected Partitions:  1..5
(15 rows)

-- View the partition-wise join plan. The plan removes the Streaming operator, eliminating the need to redistribute data between threads. This reduces data transfer overhead and improves join performance. Since both the distribution and partition keys of the hash_part_diff_key table are subsets of the join clause and maintain a one-to-one correspondence, a full partition-wise join is achievable.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM hash_part_diff_key t1, hash_part_diff_key t2 WHERE t1.a = t2.a AND t1.b = t2.b;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join (Partition-wise Join)
               Hash Cond: ((t1.a = t2.a) AND (t1.b = t2.b))
               ->  Partition Iterator
                     Iterations: 5
                     ->  Partitioned Seq Scan on hash_part_diff_key t1
                           Selected Partitions:  1..5
               ->  Hash
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part_diff_key t2
                                 Selected Partitions:  1..5
(15 rows)

-- View the partition-wise join plan. The plan removes the Streaming operator, eliminating the need to redistribute data between threads. This reduces data transfer overhead and improves join performance. Since both the distribution and partition keys of the range_part_diff_key table are subsets of the join clause and maintain a one-to-one correspondence, and each slice and partition share a consistent range, a full partition-wise join is achievable.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM range_part_diff_key t1, range_part_diff_key t2 WHERE t1.a = t2.a AND t1.b = t2.b;
                               QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join (Partition-wise Join)
               Hash Cond: ((t1.a = t2.a) AND (t1.b = t2.b))
               ->  Partition Iterator
                     Iterations: 5
                     ->  Partitioned Seq Scan on range_part_diff_key t1
                           Selected Partitions:  1..5
               ->  Hash
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on range_part_diff_key t2
                                 Selected Partitions:  1..5
(15 rows)

-- Partition-wise join is no longer supported on intermediate result tables in range distribution scenarios.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM range_part_diff_key t1, range_part_diff_key t2 ,
range_part_diff_key t3 WHERE t1.a = t2.a AND t1.b = t2.b AND  t1.a = t3.a AND t1.b = t3.b;
                                 QUERY PLAN
-----------------------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join
               Hash Cond: ((t1.a = t3.a) AND (t1.b = t3.b))
               ->  Streaming(type: LOCAL REDISTRIBUTE dop: 5/5)
                     Spawn on: All datanodes
                     ->  Hash Join (Partition-wise Join)
                           Hash Cond: ((t1.a = t2.a) AND (t1.b = t2.b))
                           ->  Partition Iterator
                                 Iterations: 5
                                 ->  Partitioned Seq Scan on range_part_diff_key t1
                                       Selected Partitions:  1..5
                           ->  Hash
                                 ->  Partition Iterator
                                       Iterations: 5
                                       ->  Partitioned Seq Scan on range_part_diff_key t2
                                             Selected Partitions:  1..5
               ->  Hash
                     ->  Streaming(type: LOCAL REDISTRIBUTE dop: 5/5)
                           Spawn on: All datanodes
                           ->  Partition Iterator
                                 Iterations: 5
                                 ->  Partitioned Seq Scan on range_part_diff_key t3
                                       Selected Partitions:  1..5
(26 rows)

-- Drop the partitioned table.
gaussdb=# DROP TABLE hash_part;
gaussdb=# DROP TABLE hash_part_diff_key;
gaussdb=# DROP TABLE range_part_diff_key;
```

NOTE

- Only partition-wise join plans in SMP scenarios display a partition-wise join message to the right of the Join operator. This message is not displayed in non-SMP scenarios.
- The execution performance of partition-wise join plans depends on the execution performance of the partition with the largest data volume. Therefore, severe data skew between partitions or partition pruning can degrade the performance of partition-wise joins.

7.8.3.5.3 Partial Partition-wise Join in SMP Scenarios

Partial partition-wise join means that one of the two joined tables is partitioned, and the other can be of any type. The system adds a Stream Redistribute operator above the table of any type to distribute its data and match it with the partitioned table. A partial partition-wise join path is generated only when the partition key of the partitioned table is a join key.

Specifications

This feature retains the full partition-wise join specifications present in SMP scenarios.

Limitations

- Partial partition-wise join cannot operate on intermediate result tables in range distribution scenarios.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- This feature retains the full partition-wise join limitations present in SMP scenarios.

Examples

In non-multi-primary mode:

```
-- Create a hash partitioned table where the distribution key and partition key are the same.
gaussdb=# CREATE TABLE hash_part
(
    a INTEGER,
    b INTEGER,
    c INTEGER
)
DISTRIBUTE BY HASH(a)
PARTITION BY HASH(a)
(
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4,
    PARTITION p5
);
CREATE TABLE

-- Set query_dop to 5 and enable SMP.
gaussdb=# SET query_dop = 5;
SET

-- Disable the nested-loop plan.
gaussdb=# SET enable_material=off;
SET
gaussdb=# SET enable_broadcast=off;
SET
gaussdb=# SET enable_nestloop=off;
SET

-- Insert 100,000 basic test data records. (a, b, and c are random integers to ensure the randomness of hash distribution.)
gaussdb=# INSERT INTO hash_part (a, b, c)
SELECT
  FLOOR(RANDOM() * 1000000)::INTEGER AS a,
  FLOOR(RANDOM() * 500000)::INTEGER AS b,
  FLOOR(RANDOM() * 1000)::INTEGER AS c
FROM GENERATE_SERIES(1, 100000);
INSERT 0 100000

-- Repeat five times, doubling the data volume each time. The final total data volume reaches 3,200,000 (100,000 × 2^5).
gaussdb=# DO $$
BEGIN
  FOR i IN 1..5 LOOP
    INSERT INTO hash_part (a, b, c)
    SELECT a, b, c FROM hash_part;
  END LOOP;
END
$$;
ANONYMOUS BLOCK EXECUTE

-- Analyze data.
gaussdb=# ANALYZE;
ANALYZE

-- Disable partition-wise join in SMP scenarios.
gaussdb=# SET enable_smp_partitionwise = off;
SET

-- View the non-partition-wise join plan. The plan shows that after two layers of operators (Partition
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Iterator and Partitioned Seq Scan) complete data scans, the Streaming(type: LOCAL REDISTRIBUTE) and
Streaming(type: SPLIT REDISTRIBUTE) operators redistribute matched data to the same Join operator.
gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM hash_part t1 INNER JOIN hash_part t2 ON (t1.a = t2.b);
                          QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join
               Hash Cond: (t2.b = t1.a)
               ->  Streaming(type: SPLIT REDISTRIBUTE dop: 5/5)
                     Spawn on: All datanodes
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part t2
                                 Selected Partitions:  1..5
               ->  Hash
                     ->  Streaming(type: LOCAL REDISTRIBUTE dop: 5/5)
                           Spawn on: All datanodes
                           ->  Partition Iterator
                                 Iterations: 5
                                 ->  Partitioned Seq Scan on hash_part t1
                                       Selected Partitions:  1..5
(19 rows)

-- Enable partition-wise join in SMP scenarios.
gaussdb=# SET enable_smp_partitionwise = on;
SET

-- View the partition-wise join plan. The partial partition-wise join plan removes the Streaming operator
from the partitioned table hash_part, eliminating the need to redistribute data between threads for that
table. This reduces data transfer overhead and improves join performance.
gaussdb=# EXPLAIN (costs off) SELECT * FROM hash_part t1, hash_part t2 WHERE t1.a=t2.b;
                          QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Streaming(type: LOCAL GATHER dop: 1/5)
         Spawn on: All datanodes
         ->  Hash Join (Partition-wise Join)
               Hash Cond: (t2.b = t1.a)
               ->  Streaming(type: SPLIT REDISTRIBUTE dop: 5/5)
                     Spawn on: All datanodes
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part t2
                                 Selected Partitions:  1..5
               ->  Hash
                     ->  Partition Iterator
                           Iterations: 5
                           ->  Partitioned Seq Scan on hash_part t1
                                 Selected Partitions:  1..5
(17 rows)

-- Drop the partitioned table.
gaussdb=# DROP TABLE hash_part;
```

NOTE

The execution performance of partition-wise join plans depends on the execution performance of the partition with the largest data volume. Therefore, severe data skew between partitions or partition pruning can degrade the performance of partition-wise joins.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.4 Partitioned Table O&M

Partitioned table O&M includes managing partitions, partitioned tables, and partitioned indexes, as well as performing concurrent operations on partitioned tables.

- Partition management: also known as partition-level DDL operations, including ADD, DROP, EXCHANGE, TRUNCATE, SPLIT, MERGE, MOVE, and RENAME.

  NOTICE

  - Operations that change the number of hash partitions—such as ADD, DROP, SPLIT, and MERGE—will cause data reshuffling. Therefore, GaussDB does not support these operations on hash partitioned tables.
  - Operations involving partition data changes—such as DROP, EXCHANGE, TRUNCATE, SPLIT, and MERGE—will make global indexes unusable. To address this issue, you can update global indexes synchronously by using the UPDATE GLOBAL INDEX clause.
  - Most DDL statements allow you to specify a partition using PARTITION/SUBPARTITION or PARTITION FOR/SUBPARTITION FOR. Use PARTITION/SUBPARTITION to specify a partition name, and use PARTITION FOR/SUBPARTITION FOR to specify a partition value within the partition range. For example, if the range of the **part1** partition is defined as [100,200), **partition part1** is equivalent to **partition for(150)**.
  - DDL execution costs vary depending on the partitions involved. The target partitions will be locked during DDL execution. Therefore, you need to evaluate the costs and impact on services beforehand. Generally, SPLIT and MERGE operations incur much higher execution costs than other DDL operations, and their costs increase proportionally with the size of the source partition. EXCHANGE costs mainly come from rebuilding and validating global indexes. MOVE costs are limited by disk I/O. Other partition DDL operations have low execution costs.
  - Statements with a specified partition do not support global index scans.

- Partitioned table management: In addition to the features inherited from ordinary tables, partitioned tables allow you to enable or disable row movement.
- Partitioned index management: You can set indexes or index partitions as unusable, and rebuild the unusable indexes or index partitions. For example, partition management operations may make global indexes unusable.
- Concurrency control on partitioned tables: In non-multi-primary mode, cross-partition DDL and DQL/DML operations cannot be executed concurrently on distributed partitioned tables, as the DDL operations lock the entire table.

7.8.4.1 ADD PARTITION

You can add partitions to an existing partitioned table to maintain new services. Each table can accommodate a maximum of 1,048,575 partitions and subpartitions in total. In the default setup, it is advisable to keep the number of partitions per table at 400 or less. For details, see **Creating Partitioned Tables**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- **ADD PARTITION** cannot work on hash partitions.
- In non-multi-primary mode, **ADD PARTITION** does not inherit sparsely partitioned indexes from the table. However, if a sparsely partitioned index is created on a partition of a subpartitioned table, any new subpartitions subsequently added to that partition will inherit the sparsely partitioned index.

7.8.4.1.1 Adding a Partition to a Range Partitioned Table

You can run the **ALTER TABLE ADD PARTITION** command to add a partition at the end of an existing partitioned table. The upper limit of the new partition must be greater than that of the original last partition.

For example, add a partition to a range partitioned table named **range_sales**.

```
ALTER TABLE range_sales ADD PARTITION date_202005 VALUES LESS THAN ('2020-06-01') TABLESPACE tb1;
```

NOTICE

If the range partitioned table already has a **MAXVALUE** partition, you cannot add any partitions to it. However, you can run the **ALTER TABLE SPLIT PARTITION** command to split existing partitions. Partition splitting also applies to scenarios where you need to add partitions before or in the middle of an existing partitioned table. For details, see **Splitting a Partition in a Range Partitioned Table**.

7.8.4.1.2 Adding a Partition to a List Partitioned Table

You can run the **ALTER TABLE ADD PARTITION** command to add a partition to a list partitioned table. Ensure that the enumerated values in the new partition do not overlap with those in existing partitions.

For example, add a partition to a list partitioned table named **list_sales**.

```
ALTER TABLE list_sales ADD PARTITION channel5 VALUES ('X') TABLESPACE tb1;
```

NOTICE

If the list partitioned table has the **DEFAULT** partition, you cannot add any partitions to it. However, you can run the **ALTER TABLE SPLIT PARTITION** command to split existing partitions.

7.8.4.1.3 Adding a Partition to a Subpartitioned Table

You can run the **ALTER TABLE ADD PARTITION** command to add a partition to a subpartitioned table. This command works when the table uses the range or list partitioning policy. If a subpartition definition is declared under the new partition, the database creates the corresponding subpartition based on the definition. If no subpartition definition is declared under the new partition, the database automatically creates a default subpartition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For example, add a partition to the subpartitioned table **range_list_sales** and create four subpartitions under it.

```
ALTER TABLE range_list_sales ADD PARTITION date_202005 VALUES LESS THAN ('2020-06-01')
TABLESPACE tb1
(
    SUBPARTITION date_202005_channel1 VALUES ('0', '1', '2'),
    SUBPARTITION date_202005_channel2 VALUES ('3', '4', '5') TABLESPACE tb2,
    SUBPARTITION date_202005_channel3 VALUES ('6', '7'),
    SUBPARTITION date_202005_channel4 VALUES ('8', '9')
);
```

Alternatively, add only a partition to the subpartitioned table **range_list_sales**.

```
ALTER TABLE range_list_sales ADD PARTITION date_202005 VALUES LESS THAN ('2020-06-01')
TABLESPACE tb1;
```

It is equivalent to the following SQL statement:

```
ALTER TABLE range_list_sales ADD PARTITION date_202005 VALUES LESS THAN ('2020-06-01')
TABLESPACE tb1
(
    SUBPARTITION date_202005_channel1 VALUES (DEFAULT)
);
```

NOTICE

When the subpartitioned table uses the hash partitioning policy, you cannot add partitions to it by running the **ALTER TABLE ADD PARTITION** command.

7.8.4.1.4 Adding a Subpartition to a Subpartitioned Table

You can run the **ALTER TABLE MODIFY PARTITION ADD SUBPARTITION** command to add a subpartition to a subpartitioned table. This command works when the table uses the range or list subpartitioning policy.

For example, add a subpartition named **date_202004** to the subpartitioned table **range_list_sales**.

```
ALTER TABLE range_list_sales MODIFY PARTITION date_202004 ADD SUBPARTITION date_202004_channel5
VALUES ('X') TABLESPACE tb2;
```

NOTICE

If the subpartitioning policy of a subpartitioned table is hash, you cannot add subpartitions by running the **ALTER TABLE MODIFY PARTITION ADD SUBPARTITION** command.

7.8.4.2 DROP PARTITION

You can run the **DROP PARTITION** command to remove unnecessary partitions. You can specify a partition name or partition value when running **DROP PARTITION**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- **DROP PARTITION** cannot work on hash partitions.
- **DROP PARTITION** will make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause to update the global indexes automatically, or rebuild the indexes manually.
- In non-multi-primary mode, if a sparsely partitioned index belongs exclusively to a partition, dropping that partition also drops the index automatically in cascading mode.

7.8.4.2.1 Dropping a Partition from a Partitioned Table

You can run the **ALTER TABLE DROP PARTITION** command to drop any partition from a specific partitioned table. This command works on range and list partitioned tables.

For example, specify a partition name to drop the **date_202005** partition from the range partitioned table **range_sales** and update the global indexes.

```
ALTER TABLE range_sales DROP PARTITION date_202005 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **'2020-05-08'** to drop the corresponding partition from the range partitioned table **range_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_sales DROP PARTITION FOR ('2020-05-08');
```

NOTICE

- When the partitioned table includes only one partition, you cannot drop the partition by running the **ALTER TABLE DROP PARTITION** command.
- Hash partitioned tables do not support dropping partitions by running the **ALTER TABLE DROP PARTITION** command.

7.8.4.2.2 Dropping a Partition from a Subpartitioned Table

You can run the **ALTER TABLE DROP PARTITION** command to drop a partition from a subpartitioned table. This command works when the table uses the range or list partitioning policy. The database will drop the partition along with all its subpartitions.

For example, specify a partition name to drop the **date_202005** partition from the subpartitioned table **range_list_sales** and update the global indexes.

```
ALTER TABLE range_list_sales DROP PARTITION date_202005 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **('2020-05-08')** to drop the corresponding partition from the subpartitioned table **range_list_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_list_sales DROP PARTITION FOR ('2020-05-08');
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- When the subpartitioned table includes only one partition, you cannot drop the partition by running the **ALTER TABLE DROP PARTITION** command.
- When the subpartitioned table uses the hash partitioning policy, you cannot drop partitions from it by running the **ALTER TABLE DROP PARTITION** command.

7.8.4.2.3 Dropping a Subpartition from a Subpartitioned Table

You can run the **ALTER TABLE DROP SUBPARTITION** command to drop a subpartition from a subpartitioned table. This command works when the table uses the range or list subpartitioning policy.

For example, specify a partition name to drop the **date_202005_channel1** subpartition from the subpartitioned table **range_list_sales** and update the global indexes.

```
ALTER TABLE range_list_sales DROP SUBPARTITION date_202005_channel1 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **('2020-05-08', '0')** to drop the corresponding subpartition from the subpartitioned table **range_list_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_list_sales DROP SUBPARTITION FOR ('2020-05-08', '0');
```

NOTICE

- When the target partition to be dropped from the subpartitioned table includes only one subpartition, you cannot drop the subpartition by running the **ALTER TABLE DROP SUBPARTITION** command.
- When the subpartitioned table uses the hash subpartitioning policy, you cannot drop subpartitions from it by running the **ALTER TABLE DROP SUBPARTITION** command.

7.8.4.3 EXCHANGE PARTITION

The **EXCHANGE PARTITION** command allows you to exchange data between a partition and an ordinary table. This enables fast data import into or export from a partitioned table, achieving efficient data loading. During service migration, **EXCHANGE PARTITION** runs significantly faster than a standard import operation. You can specify a partition name or partition value when running **EXCHANGE PARTITION**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- **EXCHANGE PARTITION** will make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause to update the global indexes automatically, or rebuild the indexes manually.
- When exchanging partitions, you can declare WITH/WITHOUT VALIDATION to specify whether to validate that ordinary table data meets the partition key constraints of the target partition. By default, the database performs this validation, which can introduce significant overhead. If you are confident that the exchanged data belongs to the target partition, declare WITHOUT VALIDATION to improve exchange performance.
- You can declare WITH VALIDATION VERBOSE. In this mode, the database validates each row of the ordinary table, inserts any data that violates the partition key constraints of the target partition to other partitions, and then exchanges the ordinary table with the target partition.

For example, consider the following partition definitions and data distribution of the ordinary table **exchange_sales**. When you exchange the **DATE_202001** partition with the **exchange_sales** table, the database will exhibit the following behaviors depending on the declared clause:

- If WITHOUT VALIDATION is declared, all data is exchanged to the **DATE_202001** partition. Because **'2020-02-03'** and **'2020-04-08'** fall outside the range of the **DATE_202001** partition, subsequent services may encounter exceptions.
- If WITH VALIDATION is declared, because **'2020-02-03'** and **'2020-04-08'** fall outside the range of the **DATE_202001** partition, the database reports an error.
- If WITH VALIDATION VERBOSE is declared, the database inserts **'2020-02-03'** into the **DATE_202002** partition, inserts **'2020-04-08'** into the **DATE_202004** partition, and exchanges the remaining data with the **DATE_202001** partition.

```
-- Partition definitions
PARTITION DATE_202001 VALUES LESS THAN ('2020-02-01'),
PARTITION DATE_202002 VALUES LESS THAN ('2020-03-01'),
PARTITION DATE_202003 VALUES LESS THAN ('2020-04-01'),
PARTITION DATE_202004 VALUES LESS THAN ('2020-05-01')
-- Data distribution of exchange_sales
('2020-01-15', '2020-01-17', '2020-01-23', '2020-02-03', '2020-04-08')
```

NOTICE

If the data to be exchanged does not fully belong to the target partition, do not declare WITHOUT VALIDATION. Otherwise, the partition constraints will be violated, causing abnormal results in subsequent DML operations on the partitioned table.

The ordinary table and partition involved in data exchange must meet the following requirements:

- The ordinary table and partition must have the same number of columns, with corresponding columns containing identical information.
- The compression settings for both the ordinary table and the partitioned table must match.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The ordinary table must have the same number of indexes as the partition's local indexes, and the index details must be identical.
- The number and details of constraints on the ordinary table and partition must be consistent.
- The ordinary table is not a temporary table.
- Neither the ordinary table nor the partitioned table can have dynamic data masking or row-level security enabled.

7.8.4.3.1 Exchanging a Partition in a Partitioned Table

You can run the **ALTER TABLE EXCHANGE PARTITION** command to exchange partitions in a partitioned table.

For example, specify a partition name to exchange the **date_202001** partition in the range partitioned table **range_sales** with the ordinary table **exchange_sales** and update the global indexes (without partition validation).

```
ALTER TABLE range_sales EXCHANGE PARTITION (date_202001) WITH TABLE exchange_sales WITHOUT VALIDATION UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **'2020-01-08'** to exchange the corresponding partition in the range partitioned table **range_sales** with the ordinary table **exchange_sales** by enabling partition validation and inserting any data that violates the target partition's constraints into other partitions in the partitioned table. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_sales EXCHANGE PARTITION FOR ('2020-01-08') WITH TABLE exchange_sales WITH VALIDATION VERBOSE;
```

7.8.4.3.2 Exchanging a Subpartition in a Subpartitioned Table

You can run the **ALTER TABLE EXCHANGE SUBPARTITION** command to exchange subpartitions in a subpartitioned table.

For example, specify a partition name to exchange the **date_202001_channel1** subpartition in the subpartitioned table **range_list_sales** with the ordinary table **exchange_sales** and update the global indexes (without partition validation).

```
ALTER TABLE range_list_sales EXCHANGE SUBPARTITION (date_202001_channel1) WITH TABLE exchange_sales WITHOUT VALIDATION UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **('2020-01-08', '0')** to exchange the corresponding subpartition in the subpartitioned table **range_list_sales** with the ordinary table **exchange_sales** by enabling partition validation and inserting any data that violates the target partition's constraints into other partitions in the partitioned table. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_list_sales EXCHANGE SUBPARTITION FOR ('2020-01-08', '0') WITH TABLE exchange_sales WITH VALIDATION VERBOSE;
```

NOTICE

Partitions in a subpartitioned table cannot be exchanged.

7.8.4.4 TRUNCATE PARTITION

You can run the **TRUNCATE PARTITION** command to quickly clear data from a partition. This command works like **DROP PARTITION**, but it deletes only the data

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

while keeping the partition's definition and physical files intact. You can specify a partition name or partition value when running **TRUNCATE PARTITION**.

NOTICE

TRUNCATE PARTITION will make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause to update the global indexes automatically, or rebuild the indexes manually.

7.8.4.4.1 Clearing a Partition in a Partitioned Table

You can run the **ALTER TABLE TRUNCATE PARTITION** command to clear any partition in a specified partitioned table.

For example, specify a partition name to clear the **date_202005** partition in the range partitioned table **range_sales** and update the global indexes.

```
ALTER TABLE range_sales TRUNCATE PARTITION date_202005 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **'2020-05-08'** to clear the corresponding partition from the range partitioned table **range_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_sales TRUNCATE PARTITION FOR ('2020-05-08');
```

7.8.4.4.2 Clearing a Partition in a Subpartitioned Table

You can run the **ALTER TABLE TRUNCATE PARTITION** command to clear a partition in a subpartitioned table. The database then clears all subpartitions within that partition.

For example, specify a partition name to clear the **date_202005** partition in the subpartitioned table **range_list_sales** and update the global indexes.

```
ALTER TABLE range_list_sales TRUNCATE PARTITION date_202005 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **('2020-05-08')** to clear the corresponding partition from the subpartitioned table **range_list_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_list_sales TRUNCATE PARTITION FOR ('2020-05-08');
```

7.8.4.4.3 Clearing a Subpartition in a Subpartitioned Table

You can run the **ALTER TABLE TRUNCATE SUBPARTITION** command to clear a subpartition in a subpartitioned table.

For example, specify a partition name to clear the **date_202005_channel1** subpartition in the subpartitioned table **range_list_sales** and update the global indexes.

```
ALTER TABLE range_list_sales TRUNCATE SUBPARTITION date_202005_channel1 UPDATE GLOBAL INDEX;
```

Alternatively, specify the partition value **('2020-05-08', '0')** to clear the corresponding subpartition from the subpartitioned table **range_list_sales**. The absence of the UPDATE GLOBAL INDEX clause means that global indexes will become unusable after you run this command.

```
ALTER TABLE range_list_sales TRUNCATE SUBPARTITION FOR ('2020-05-08', '0');
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.4.5 SPLIT PARTITION

You can run the **SPLIT PARTITION** command to split a partition into two or more new partitions. Consider this operation when partition data grows too large or when you need to add a partition to a range partitioned table with **MAXVALUE** or a list partitioned table with **DEFAULT**. You can specify a split point to split a partition into two; if you omit this point, the command will split the partition into multiple partitions. You can specify a partition name or partition value when running **SPLIT PARTITION**.

NOTICE

- **SPLIT PARTITION** cannot work on hash partitions.
- **SPLIT PARTITION** will make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause to update the global indexes automatically, or rebuild the indexes manually.
- If the target partition contains a sparsely partitioned index, the partition cannot be split. However, if a sparsely partitioned index exists on a partition of a subpartitioned table, you can split its subpartitions.
- You can name the new partitions the same as the source partition. For example, you can split **p1** into **p1** and **p2**. However, the database treats partitions with the same name before and after the split as distinct partitions. This distinction affects queries on the original **p1** partition during the split. For details, see **DQL/DML-DDL Concurrency**.

7.8.4.5.1 Splitting a Partition in a Range Partitioned Table

You can run the **ALTER TABLE SPLIT PARTITION** command to split a partition in a range partitioned table.

The following example assumes that the **date_202001** partition in the range partitioned table **range_sales** is defined with the range ['2020-01-01','2020-02-01'). It specifies the split point **'2020-01-16'** to split the **date_202001** partition into two partitions and update the global indexes.

```
ALTER TABLE range_sales SPLIT PARTITION date_202001 AT ('2020-01-16') INTO
(
    PARTITION date_202001_p1, -- The upper limit of the first partition is '2020-01-16'.
    PARTITION date_202001_p2  -- The upper limit of the second partition is '2020-02-01'.
) UPDATE GLOBAL INDEX;
```

Alternatively, specify no split point in order to split the **date_202001** partition into multiple partitions and update the global indexes.

```
ALTER TABLE range_sales SPLIT PARTITION date_202001 INTO
(
    PARTITION date_202001_p1 VALUES LESS THAN ('2020-01-11'), -- The upper limit of the first partition is '2020-01-11'.
    PARTITION date_202001_p2 VALUES LESS THAN ('2020-01-21'), -- The upper limit of the second partition is '2020-01-21'.
    PARTITION date_202001_p3 -- The upper limit of the third partition is '2020-02-01'.
)UPDATE GLOBAL INDEX;
```

Alternatively, specify a partition value rather than a name to split the corresponding partition.

```
ALTER TABLE range_sales SPLIT PARTITION FOR ('2020-01-15') AT ('2020-01-16') INTO
(
    PARTITION date_202001_p1, -- The upper limit of the first partition is '2020-01-16'.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   PARTITION date_202001_p2  -- The upper limit of the second partition is '2020-02-01'.
) UPDATE GLOBAL INDEX;
```

NOTICE

To split the **MAXVALUE** partition, do not declare the **MAXVALUE** range in earlier partitions. The last partition automatically inherits the **MAXVALUE** range.

7.8.4.5.2 Splitting a Partition in a List Partitioned Table

You can run the **ALTER TABLE SPLIT PARTITION** command to split a partition in a list partitioned table.

The following example assumes that the **channel2** partition in the list partitioned table **list_sales** is defined with the range ('6', '7', '8', '9'). It specifies the split point **('6', '7')** to split the **channel2** partition into two partitions and update the global indexes.

```
ALTER TABLE list_sales SPLIT PARTITION channel2 VALUES ('6', '7') INTO
(
   PARTITION channel2_1, -- The range of the first partition is ('6', '7').
   PARTITION channel2_2  -- The range of the second partition is ('8', '9').
) UPDATE GLOBAL INDEX;
```

Alternatively, specify no split point in order to split the **channel2** partition into multiple partitions and update the global indexes.

```
ALTER TABLE list_sales SPLIT PARTITION channel2 INTO
(
   PARTITION channel2_1 VALUES ('6'), -- The range of the first partition is ('6').
   PARTITION channel2_2 VALUES ('8'), -- The range of the second partition is ('8').
   PARTITION channel2_3 -- The range of the third partition is ('7', '9').
)UPDATE GLOBAL INDEX;
```

Alternatively, specify a partition value rather than a name to split the corresponding partition.

```
ALTER TABLE list_sales SPLIT PARTITION FOR ('6') VALUES ('6', '7') INTO
(
   PARTITION channel2_1, -- The range of the first partition is ('6', '7').
   PARTITION channel2_2  -- The range of the second partition is ('8', '9').
) UPDATE GLOBAL INDEX;
```

NOTICE

To split the **DEFAULT** partition, do not declare the **DEFAULT** range in earlier partitions. The last partition automatically inherits the **DEFAULT** range.

7.8.4.5.3 Splitting a Subpartition in a *-Range Subpartitioned Table

You can run the **ALTER TABLE SPLIT SUBPARTITION** command to split a subpartition in a *-range subpartitioned table.

The following example assumes that the **channel1_customer4** subpartition in the *-range subpartitioned table **list_range_sales** is defined with the range [1000,MAXVALUE). It specifies the split point **1200** to split the **channel1_customer4** subpartition into two subpartitions and update the global indexes.

```
ALTER TABLE list_range_sales SPLIT SUBPARTITION channel1_customer4 AT (1200) INTO
(
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   SUBPARTITION channel1_customer4_p1, -- The upper limit of the first subpartition is 1200.
   SUBPARTITION channel1_customer4_p2  -- The upper limit of the second subpartition is MAXVALUE.
) UPDATE GLOBAL INDEX;
```

Alternatively, specify no split point in order to split the **channel1_customer4** subpartition into multiple subpartitions and update the global indexes.

```
ALTER TABLE list_range_sales SPLIT SUBPARTITION channel1_customer4 INTO
(
   SUBPARTITION channel1_customer4_p1 VALUES LESS THAN (1200), -- The upper limit of the first subpartition is 1200.
   SUBPARTITION channel1_customer4_p2 VALUES LESS THAN (1400), -- The upper limit of the second subpartition is 1400.
   SUBPARTITION channel1_customer4_p3 -- The upper limit of the third subpartition is MAXVALUE.
)UPDATE GLOBAL INDEX;
```

Alternatively, specify a subpartition value rather than a name to split the corresponding subpartition.

```
ALTER TABLE list_range_sales SPLIT SUBPARTITION FOR ('1', 1200) AT (1200) INTO
(
   PARTITION channel1_customer4_p1, -- The upper limit of the first subpartition is 1200.
   PARTITION channel1_customer4_p2  -- The upper limit of the second subpartition is MAXVALUE.
) UPDATE GLOBAL INDEX;
```

NOTICE

To split the **MAXVALUE** subpartition, do not declare the **MAXVALUE** range in earlier subpartitions. The last subpartition automatically inherits the **MAXVALUE** range.

7.8.4.5.4 Splitting a Subpartition in a *-List Subpartitioned Table

You can run the **ALTER TABLE SPLIT SUBPARTITION** command to split a subpartition in a *-list subpartitioned table.

The following example assumes that the **product2_channel2** subpartition in the *-list subpartitioned table **hash_list_sales** is defined with the **DEFAULT** range. It specifies a split point to split the subpartition into two subpartitions and update the global indexes.

```
ALTER TABLE hash_list_sales SPLIT SUBPARTITION product2_channel2 VALUES ('6', '7', '8', '9') INTO
(
   SUBPARTITION product2_channel2_p1, -- The range of the first subpartition is ('6', '7', '8', '9').
   SUBPARTITION product2_channel2_p2  -- The range of the second subpartition is DEFAULT.
) UPDATE GLOBAL INDEX;
```

Alternatively, specify no split point in order to split the **product2_channel2** subpartition into multiple subpartitions and update the global indexes.

```
ALTER TABLE hash_list_sales SPLIT SUBPARTITION product2_channel2 INTO
(
   SUBPARTITION product2_channel2_p1 VALUES ('6', '7', '8'), -- The range of the first subpartition is ('6', '7', '8').
   SUBPARTITION product2_channel2_p2 VALUES ('9', '10'), -- The range of the second subpartition is ('9', '10').
   SUBPARTITION product2_channel2_p3 -- The range of the third subpartition is DEFAULT.
) UPDATE GLOBAL INDEX;
```

Alternatively, specify a subpartition value rather than a name to split the corresponding subpartition.

```
ALTER TABLE hash_list_sales SPLIT SUBPARTITION FOR (1200, '6') VALUES ('6', '7', '8', '9') INTO
(
   SUBPARTITION product2_channel2_p1, -- The range of the first subpartition is ('6', '7', '8', '9').
   SUBPARTITION product2_channel2_p2  -- The range of the second subpartition is DEFAULT.
) UPDATE GLOBAL INDEX;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

To split the **DEFAULT** subpartition, do not declare the **DEFAULT** range in earlier subpartitions. The last subpartition automatically inherits the **DEFAULT** range.

7.8.4.6 MERGE PARTITION

You can run the **MERGE PARTITION** command to merge multiple partitions into a new partition. When running **MERGE PARTITION**, specify a partition name but not a partition value.

NOTICE

- **MERGE PARTITION** cannot work on hash partitions.
- **MERGE PARTITION** will make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause to update the global indexes automatically, or rebuild the indexes manually.
- You cannot merge partitions that contain sparsely partitioned indexes. However, if a sparsely partitioned index exists on a partition of a subpartitioned table, you can merge its subpartitions.
- When merging range partitions, you can name the new partition after the last source partition. For example, merge **p1** and **p2** into **p2**. When merging list partitions, you can name the new partition after any source partition. For example, merge **p1** and **p2** into **p1**.
- If the new partition shares the same name as a source partition, the database treats it as an inheritance of that source partition.

7.8.4.6.1 Merging Partitions in a Partitioned Table

You can run the **ALTER TABLE MERGE PARTITIONS** command to merge multiple partitions into one.

For example, merge the **date_202001** and **date_202002** partitions in the range partitioned table **range_sales** into a new partition and update the global indexes.

```
ALTER TABLE range_sales MERGE PARTITIONS date_202001, date_202002 INTO
    PARTITION date_2020_old UPDATE GLOBAL INDEX;
```

7.8.4.6.2 Merging Subpartitions in a Subpartitioned Table

You can run the **ALTER TABLE MERGE SUBPARTITIONS** command to merge multiple subpartitions into one.

For example, merge the **product1_channel1**, **product1_channel2**, and **product1_channel3** subpartitions in the subpartitioned table **hash_list_sales** into a new subpartition and update the global indexes.

```
ALTER TABLE hash_list_sales MERGE SUBPARTITIONS product1_channel1, product1_channel2,
product1_channel3 INTO
    SUBPARTITION product1_channel1 UPDATE GLOBAL INDEX;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.4.7 MOVE PARTITION

You can run the **MOVE PARTITION** command to move a partition to a different tablespace. You can specify a partition name or partition value when running **MOVE PARTITION**.

7.8.4.7.1 Moving a Partition in a Partitioned Table

You can run the **ALTER TABLE MOVE PARTITION** command to move a partition in a partitioned table.

For example, specify a partition name to move the **date_202001** partition in the range partitioned table **range_sales** to the **tb1** tablespace.

```
ALTER TABLE range_sales MOVE PARTITION date_202001 TABLESPACE tb1;
```

Alternatively, specify the partition value **'0'** to move the corresponding partition in the list partitioned table **list_sales** to the **tb1** tablespace.

```
ALTER TABLE list_sales MOVE PARTITION FOR ('0') TABLESPACE tb1;
```

7.8.4.7.2 Moving a Subpartition in a Subpartitioned Table

You can run the **ALTER TABLE MOVE SUBPARTITION** command to move a subpartition in a subpartitioned table.

For example, specify a partition name to move the **date_202001_channel1** subpartition in the subpartitioned table **range_list_sales** to the **tb1** tablespace.

```
ALTER TABLE range_list_sales MOVE SUBPARTITION date_202001_channel1 TABLESPACE tb1;
```

Alternatively, specify the partition value **('2020-01-08', '0')** to move the corresponding subpartition in the subpartitioned table **range_list_sales** to the **tb1** tablespace.

```
ALTER TABLE range_list_sales MOVE SUBPARTITION FOR ('2020-01-08', '0') TABLESPACE tb1;
```

7.8.4.8 RENAME PARTITION

You can run the **RENAME PARTITION** command to assign a new name to a partition. You can specify a partition name or partition value when running **RENAME PARTITION**.

7.8.4.8.1 Renaming a Partition in a Partitioned Table

You can run the **ALTER TABLE RENAME PARTITION** command to rename a partition in a partitioned table.

For example, specify a partition name to rename the **date_202001** partition in the range partitioned table **range_sales**.

```
ALTER TABLE range_sales RENAME PARTITION date_202001 TO date_202001_new;
```

Alternatively, specify the partition value **'0'** to rename the corresponding partition in the list partitioned table **list_sales**.

```
ALTER TABLE list_sales RENAME PARTITION FOR ('0') TO channel_new;
```

7.8.4.8.2 Renaming a Partition in a Subpartitioned Table

You can run the **ALTER TABLE RENAME PARTITION** command to rename a partition in a subpartitioned table.

The command works the same way as it does for a partitioned table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.4.8.3 Renaming a Subpartition in a Subpartitioned Table

You can run the **ALTER TABLE RENAME SUBPARTITION** command to rename a subpartition in a subpartitioned table.

For example, specify a partition name to rename the **date_202001_channel1** subpartition in the subpartitioned table **range_list_sales**.

```
ALTER TABLE range_list_sales RENAME SUBPARTITION date_202001_channel1 TO date_202001_channelnew;
```

Alternatively, specify the partition value **('2020-01-08', '0')** to rename the corresponding subpartition in the subpartitioned table **range_list_sales**.

```
ALTER TABLE range_list_sales RENAME SUBPARTITION FOR ('2020-01-08', '0') TO date_202001_channelnew;
```

7.8.4.8.4 Renaming the Index Partition for a Local Index

You can run the **ALTER INDEX RENAME PARTITION** command to rename the index partition for a local index.

The command works the same way as renaming a partition in a partitioned table.

7.8.4.9 ROW MOVEMENT

You can run the **ALTER TABLE ENABLE/DISABLE ROW MOVEMENT** command to enable or disable row movement for a partitioned table.

When row movement is enabled, an UPDATE operation can move data from one partition to another. When row movement is disabled, the UPDATE operation triggers a service error.

NOTICE

If your services do not allow updating partition key columns, you are advised to disable row movement.

For example, create a list partitioned table and enable row movement. In this setup, partition key columns can be updated across partitions. If row movement is disabled, updating partition key columns across partitions will trigger an error.

```
CREATE TABLE list_sales
(
   product_id     INT4 NOT NULL,
   customer_id    INT4 PRIMARY KEY,
   time_id        DATE,
   channel_id     CHAR(1),
   type_id        INT4,
   quantity_sold  NUMERIC(3),
   amount_sold    NUMERIC(10,2)
)
PARTITION BY LIST (channel_id)
(
   PARTITION channel1 VALUES ('0', '1', '2'),
   PARTITION channel2 VALUES ('3', '4', '5'),
   PARTITION channel3 VALUES ('6', '7'),
   PARTITION channel4 VALUES ('8', '9')
) ENABLE ROW MOVEMENT;
INSERT INTO list_sales VALUES (153241,65143129,'2021-05-07','0',864134,89,34);
-- The cross-partition update succeeds, and data is moved from the channel1 partition to the channel2 partition.
UPDATE list_sales SET channel_id = '3' WHERE channel_id = '0';
-- Disable row movement for the partitioned table.
ALTER TABLE list_sales DISABLE ROW MOVEMENT;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- The cross-partition update fails with this error: fail to update partitioned table "list_sales".
UPDATE list_sales SET channel_id = '0' WHERE channel_id = '3';
-- The intra-partition update still succeeds.
UPDATE list_sales SET channel_id = '4' WHERE channel_id = '3';
```

7.8.4.10 UNUSABLE/REBUILD

To stop maintaining a partitioned table index or an index partition, set it as unusable by running the **UNUSABLE** command. When you want to resume the index functionality, run the **REBUILD** command.

Certain partition-level DDL operations—such as DROP, EXCHANGE, TRUNCATE, SPLIT, and MERGE—also make global indexes unusable. To address this, you can either include the UPDATE GLOBAL INDEX clause in your DDL statements to update the global indexes automatically, or rebuild the indexes manually.

7.8.4.10.1 Setting an Index as Unusable/Rebuilding an Index

To set an index as usable or not, run the **ALTER INDEX** command.

For example, set the **range_sales_idx** index on the partitioned table **range_sales** as unusable.

```
ALTER INDEX range_sales_idx UNUSABLE;
```

Run the following command to rebuild the **range_sales_idx** index:

```
ALTER INDEX range_sales_idx REBUILD;
```

7.8.4.10.2 Setting Local Index Partitions as Unusable/Rebuilding Local Index Partitions

- To set local index partitions as usable or not, run the **ALTER INDEX PARTITION** command.
- To set all index partitions on a specified partition as unusable or not, run the **ALTER TABLE MODIFY PARTITION** command. When applied to a partition in a subpartitioned table, this command affects all subpartitions within that partition.
- To set all index partitions on a specified subpartition in a subpartitioned table as unusable or not, run the **ALTER TABLE MODIFY SUBPARTITION** command.

Assume that the partitioned table **range_sales** has two local indexes: **range_sales_idx1** and **range_sales_idx2**, corresponding to the index partitions **range_sales_idx1_part1** and **range_sales_idx2_part1** on the **date_202001** partition, respectively.

The partitioned indexes of the partitioned table can be maintained using the following example syntax:

- Run the following command to set all index partitions on the **date_202001** partition as unusable:

  ```
  ALTER TABLE range_sales MODIFY PARTITION date_202001 UNUSABLE LOCAL INDEXES;
  ```

- Alternatively, run the following command to set the index partition **range_sales_idx1_part1** on the **date_202001** partition as unusable:

  ```
  ALTER INDEX range_sales_idx1 MODIFY PARTITION range_sales_idx1_part1 UNUSABLE;
  ```

- Run the following command to rebuild all index partitions on the **date_202001** partition:

  ```
  ALTER TABLE range_sales MODIFY PARTITION date_202001 REBUILD UNUSABLE LOCAL INDEXES;
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Alternatively, run the following command to rebuild the index partition **range_sales_idx1_part1** on the **date_202001** partition:
```
ALTER INDEX range_sales_idx1 REBUILD PARTITION range_sales_idx1_part1;
```

Assume that the subpartitioned table **list_range_sales** has two local indexes: **list_range_sales_idx1** and **list_range_sales_idx2**. The table includes a partition named **channel1**, which includes the **channel1_product1**, **channel1_product2**, and **channel1_product3** subpartitions. The **channel1_product1** subpartition corresponds to the index partitions **channel1_product1_idx1** and **channel1_product1_idx2**.

The partitioned indexes of the subpartitioned table can be maintained as follows:

- Run the following command to set all index partitions on subpartitions—including **channel1_product1**, **channel1_product2**, and **channel1_product3**—within the **channel1** partition as unusable:
```
ALTER TABLE list_range_sales MODIFY PARTITION channel1 UNUSABLE LOCAL INDEXES;
```
- Run the following command to rebuild all index partitions on subpartitions within the **channel1** partition:
```
ALTER TABLE list_range_sales MODIFY PARTITION channel1 REBUILD UNUSABLE LOCAL INDEXES;
```

The subpartitioned indexes of the subpartitioned table can be maintained as follows:

- Run the following command to set all index partitions on the **channel1_product1** subpartition as unusable:
```
ALTER TABLE list_range_sales MODIFY SUBPARTITION channel1_product1 UNUSABLE LOCAL INDEXES;
```
- Run the following command to rebuild all index partitions on the **channel1_product1** subpartition:
```
ALTER TABLE list_range_sales MODIFY SUBPARTITION channel1_product1 REBUILD UNUSABLE LOCAL INDEXES;
```
- Alternatively, run the following command to set the index partition **channel1_product1_idx1** on the **channel1_product1** subpartition as unusable:
```
ALTER INDEX list_range_sales_idx1 MODIFY PARTITION channel1_product1_idx1 UNUSABLE;
```
- Run the following command to rebuild the index partition **channel1_product1_idx1** on the **channel1_product1** subpartition:
```
ALTER INDEX list_range_sales_idx1 REBUILD PARTITION channel1_product1_idx1;
```

7.8.5 Partition Concurrency Control

Partition concurrency control sets specification limits for concurrent DQL, DML, and DDL operations on partitioned tables. Refer to this section when designing concurrent operations on partitioned tables, especially during partition maintenance.

7.8.5.1 Standard Lock Design

Partitioned tables use both table locks and partition locks. Applying standard locks at different levels ensures proper control during concurrent DQL, DML, and DDL operations on tables and partitions. **Table 7-8** shows the compatibility rules between different lock levels. Each pair of standard locks marked with '√' does not block each other and can execute in parallel.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-8 Compatibility rules between standard locks

Compatibility Rule	ACCESS_SHARE	ROW_SHARE	ROW_EXCLUSIVE	SHARE_UPDATE_EXCLUSIVE	SHARE	SHARE_ROW_EXCLUSIVE	EXCLUSIVE	ACCESS_EXCLUSIVE
ACCESS_SHARE	√	√	√	√	√	√	√	×
ROW_SHARE	√	√	√	√	√	√	×	×
ROW_EXCLUSIVE	√	√	√	√	×	×	×	×
SHARE_UPDATE_EXCLUSIVE	√	√	√	×	×	×	×	×
SHARE	√	√	×	×	√	×	×	×
SHARE_ROW_EXCLUSIVE	√	√	×	×	×	×	×	×
EXCLUSIVE	√	×	×	×	×	×	×	×
ACCESS_EXCLUSIVE	×	×	×	×	×	×	×	×

When multiple operations target the same partition in a partitioned table, the database applies table and partition locks at different levels to manage concurrent access. **Table 7-9** details the specific lock levels applied to different operations. The numbers 1 through 8 represent the eight levels of standard locks defined in **Table 7-8**: ACCESS_SHARE, ROW_SHARE, ROW_EXCLUSIVE, SHARE_UPDATE_EXCLUSIVE, SHARE, SHARE_ROW_EXCLUSIVE, EXCLUSIVE, and ACCESS_EXCLUSIVE.

Table 7-9 Lock levels applied to different operations on partitioned tables

Operation Pattern	Table Lock+Partition Lock on Partitioned Table	Table Lock+Partition Lock+Subpartition Lock on Subpartitioned Table
SELECT	1-1	1-1-1
SELECT FOR UPDATE	2-2	2-2-2

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation Pattern	Table Lock+Partition Lock on Partitioned Table	Table Lock+Partition Lock+Subpartition Lock on Subpartitioned Table
DML operations, including INSERT, UPDATE, DELETE, UPSERT, MERGE INTO, and COPY	3-3	3-3-3
Most partition-level DDL operations, including ADD, DROP, EXCHANGE, TRUNCATE, MOVE and RENAME	4-8	4-8-8 (applied to partitions in a subpartitioned table) 4-4-8 (applied to subpartitions in a subpartitioned table)
CREATE INDEX (excluding sparsely partitioned indexes) and REBUILD INDEX	5-5	5-5-5
CREATE INDEX (sparsely partitioned indexes)	3-5	3-3-5
REBUILD INDEX PARTITION	1-5	1-5-5
ANALYZE and VACUUM	4-4	4-4-4
Other DDL operations on partitioned tables, including partition-level SPLIT and MERGE	8-8	8-8-8

The database allows parallel operations only when both the table lock and partition lock applied to them comply with the compatibility rules in **Table 7-8**. If either lock is incompatible, the database blocks the parallel operations.

NOTE

The table-level lock applied by DDL operations (ADD/DROP/TRUNCATE/EXCHANGE/MOVE/RENAME) to a partitioned table is controlled by the GUC parameter **enable_partition_ddl_lowlevel_lock**. If **enable_partition_ddl_lowlevel_lock** is set to **on**, a level-4 lock is applied to the table; if it is set to **off**, a level-8 lock is applied.

7.8.5.2 DQL/DML-DQL/DML Concurrency

The database applies Level 1–3 standard locks to DQL and DML operations on tables and partitions. By design, DQL and DML operations do not block each other.

Partitioned tables support DQL/DML-DQL/DML concurrency.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.5.3 DQL/DML-DDL Concurrency

Performing table-level DDL operations or partition-level SPLIT/MERGE (DDL operations) applies a level-8 lock to the partitioned table, blocking all DQL and DML operations on that table.

Performing partition-level ADD, DROP, EXCHANGE, TRUNCATE, MOVE, or RENAME—which are DDL operations—applies a level-4 lock to the partitioned table and a level-8 lock to the target partition. DQL/DML and DDL operations can be performed concurrently when they target different partitions. However, when they target the same partition, later operations will be blocked by earlier ones.

NOTICE

- During partition-level DDL operations, avoid performing DQL/DML operations on the target partition simultaneously.
- If concurrent DDL and DQL/DML operations target overlapping partitions, serial blocking may cause the DQL/DML operations to run before or after the DDL operations. Be aware of related possible results. For example, when TRUNCATE and INSERT target the same partition: if TRUNCATE runs before INSERT, the target partition contains data after the operations complete; if TRUNCATE runs after INSERT, the target partition does not contain data.
- If the partitioned table is a bucket table with global indexes, the global indexes will be rebuilt upon three partition-level DDL operations—MERGE, EXCHANGE, or SET TABLESPACE. During this period, DML operations on the table are blocked.

DQL/DML-DDL Concurrency Across Partitions

GaussDB supports DQL/DML-DDL concurrency across partitions.

Here are some examples demonstrating concurrency support in the partitioned table **range_sales**:

```
gaussdb=# CREATE TABLE range_sales
(
    product_id      INT4 NOT NULL,
    customer_id     INT4 NOT NULL,
    time_id         DATE,
    channel_id      CHAR(1),
    type_id         INT4,
    quantity_sold   NUMERIC(3),
    amount_sold     NUMERIC(10,2)
)
PARTITION BY RANGE (time_id)
(
    PARTITION time_2008 VALUES LESS THAN ('2009-01-01'),
    PARTITION time_2009 VALUES LESS THAN ('2010-01-01'),
    PARTITION time_2010 VALUES LESS THAN ('2011-01-01'),
    PARTITION time_2011 VALUES LESS THAN ('2012-01-01')
);

gaussdb=# CREATE TABLE temp
(
    product_id      INT4 NOT NULL,
    customer_id     INT4 NOT NULL,
    time_id         DATE,
    channel_id      CHAR(1),
    type_id         INT4,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    quantity_sold  NUMERIC(3),
    amount_sold    NUMERIC(10,2)
);
```

Typically, the partitioned table supports the following concurrent operations:

```
-- Case 1: Insert the time_2011 partition and clear the time_2008 partition concurrently. These operations do not block each other.
gaussdb=# \parallel on
gaussdb=# INSERT INTO range_sales VALUES (455124, 92121433, '2011-09-17', 'X', 4513, 7, 17);
gaussdb=# ALTER TABLE range_sales TRUNCATE PARTITION time_2008 UPDATE GLOBAL INDEX;
gaussdb=# \parallel off

-- Case 2: Query the time_2010 partition and exchange the time_2009 partition concurrently. These operations do not block each other.
gaussdb=# \parallel on
gaussdb=# SELECT COUNT(*) FROM range_sales PARTITION (time_2010);
gaussdb=# ALTER TABLE range_sales EXCHANGE PARTITION (time_2009) WITH TABLE temp UPDATE GLOBAL INDEX;
gaussdb=# \parallel off

-- Case 3: Update the partitioned table range_sales and drop the time_2008 partition concurrently. These operations do not block each other, because the conditions specified in the SQL statement prune the UPDATE to the time_2010 and time_2011 partitions.
gaussdb=# \parallel on
gaussdb=# UPDATE range_sales SET channel_id = 'T' WHERE channel_id = 'X' AND time_id > '2010-06-01';
gaussdb=# ALTER TABLE range_sales DROP PARTITION time_2008 UPDATE GLOBAL INDEX;
gaussdb=# \parallel off

-- Case 4: Perform any DQL/DML operations on the partitioned table range_sales and add the time_2012 partition concurrently. These operations do not block each other, because the new partition is invisible to other operations.
gaussdb=# \parallel on
gaussdb=# DELETE FROM range_sales WHERE channel_id = 'T';
gaussdb=# ALTER TABLE range_sales ADD PARTITION time_2012 VALUES LESS THAN ('2013-01-01');
gaussdb=# \parallel off
```

DQL/DML-DDL Concurrency on the Same Partition

GaussDB does not support DQL/DML-DDL concurrency on the same partition. Later operations will be blocked by earlier ones.

In principle, you are advised not to perform DQL/DML operations on a partition while it undergoes DDL operations. This is because abrupt changes to the partition status can lead to unexpected query results.

If improper operation patterns or pruning failures cause DQL/DML and DDL operations to target overlapping partitions, consider the following two scenarios:

Scenario 1: DQL/DML operations run before DDL operations. In this scenario, the DDL operations will be blocked. Wait until the DQL/DML operations are committed.

Scenario 2: DDL operations run before DQL/DML operations. In this scenario, the DQL/DML operations will be blocked. Wait until the DDL operations are committed. Changes to partition metadata may lead to unexpected results. For data consistency, the expected results are defined according to the following rules:

- **ADD PARTITION**

  ADD PARTITION generates a new partition that remains invisible to any concurrent DQL/DML operations. Therefore, these operations proceed without blocking.

- **DROP PARTITION**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DROP PARTITION removes an existing partition. During this period, any DQL/DML operations targeting that partition will be blocked. After blocking ends, these operations skip the removed partition.

- **TRUNCATE PARTITION**

  TRUNCATE PARTITION clears data from an existing partition. During this period, any DQL/DML operations targeting that partition will be blocked. After blocking ends, these operations resume on the partition.

  Note that once the TRUNCATE operation commits, the partition contains no data, so queries on it return no results.

- **EXCHANGE PARTITION**

  EXCHANGE PARTITION exchanges an existing partition with an ordinary table. During this period, any DQL/DML operations targeting that partition will be blocked. After blocking ends, these operations resume on the partition, which now contains the data from the ordinary table.

  Exception: If the partitioned table has a global index and the **EXCHANGE PARTITION** command includes the UPDATE GLOBAL INDEX clause, partitioned table queries using the global index during this period cannot access data in the post-exchange partition. As a result, the database reports an error for these queries after blocking ends.

  ERROR: partition xxxxxx does not exist on relation "xxxxxx"

  DETAIL: this partition may have already been dropped by cocurrent DDL operations EXCHANGE PARTITION

- **SPLIT PARTITION**

  SPLIT PARTITION blocks DQL/DML operations on the entire table, including the target partition. Once SPLIT is committed, DQL/DML operations are performed based on the post-SPLIT partitioned table definition.

- **MERGE PARTITION**

  MERGE PARTITION blocks DQL/DML operations on the entire table, including the target partition. Once MERGE is committed, DQL/DML operations are performed based on the post-MERGE partitioned table definition.

- **RENAME PARTITION**

  RENAME PARTITION does not alter the partition structure. During this period, any DQL/DML operations targeting that partition will not throw exceptions but will remain blocked until the RENAME operation is committed.

- **MOVE PARTITION**

  MOVE PARTITION does not alter the partition structure. During this period, any DQL/DML operations targeting that partition will not throw exceptions but will remain blocked until the MOVE operation is committed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- During DQL/DML operations on a partition, performing multiple consecutive DDL operations on that partition may occasionally trigger an error. This error occurs because the partition cannot be found, as a DDL operation has deleted it.
- There is a low probability of deadlocks or lock timeouts when DQL/DML and DDL operations are performed concurrently on the same partition. This is because certain DQL/DML operations apply partition locks only on DNs. If the lock sequence on these DNs differs from the lock sequence of the DDL operations, a deadlock or lock timeout may occur. To prevent deadlocks, you can set the GUC parameter **enable_partition_ddl_lowlevel_lock** to **off**. However, this may impact performance. For details, see the description of the GUC parameter **enable_partition_ddl_lowlevel_lock**.

7.8.6 System Views & DFx Related to Partitioned Tables

7.8.6.1 System Views Related to Partitioned Tables

In non-multi-primary mode, the system views related to partitioned tables are classified into three types based on permissions. For specific fields, refer to "System Catalogs and System Views > System Views" in *Reference*.

- Views for all partitions:
  - ADM_PART_TABLES: stores information about all partitioned tables.
  - ADM_TAB_PARTITIONS: stores information about all partitions.
  - ADM_PART_INDEXES: stores information about all local indexes.
  - ADM_IND_PARTITIONS: stores information about all index partitions.
- Views accessible to the current user:
  - DB_PART_TABLES: stores information about partitioned tables accessible to the current user.
  - DB_TAB_PARTITIONS: stores information about partitions accessible to the current user.
  - DB_PART_INDEXES: stores information about local indexes accessible to the current user.
  - DB_IND_PARTITIONS: stores information about index partitions accessible to the current user.
- Views owned by the current user:
  - MY_PART_TABLES: stores information about partitioned tables owned by the current user.
  - MY_TAB_PARTITIONS: stores information about partitions owned by the current user.
  - MY_PART_INDEXES: stores local indexes owned by the current user.
  - MY_IND_PARTITIONS: stores information about index partitions owned by the current user.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.8.6.2 Built-in Tool Functions Related to Partitioned Tables

Information About Table Creation

- Preparation: Create a table.

```
CREATE TABLE test_range_pt (a INT, b INT, c INT)
PARTITION BY RANGE (a)
(
   PARTITION p1 VALUES LESS THAN (2000),
   PARTITION p2 VALUES LESS THAN (3000),
   partition p3 VALUES LESS THAN (4000),
   partition p4 VALUES LESS THAN (5000),
   partition p5 VALUES LESS THAN (MAXVALUE)
)ENABLE ROW MOVEMENT;
```

- Query the OID of a partitioned table.

```
SELECT oid FROM pg_class WHERE relname = 'test_range_pt';
oid
-------
49290
(1 row)
```

- Query partition information.

```
SELECT oid,relname,parttype,parentid,boundaries FROM pg_partition WHERE parentid = 49290;
oid  |    relname    | parttype | parentid | boundaries
-------+---------------+----------+----------+------------
49293 | test_range_pt   | r        |    49290 |
49294 | p1              | p        |    49290 | {2000}
49295 | p2              | p        |    49290 | {3000}
49296 | p3              | p        |    49290 | {4000}
49297 | p4              | p        |    49290 | {5000}
49298 | p5              | p        |    49290 | {NULL}
(6 rows)
```

- Create an index.

```
CREATE INDEX idx_range_a ON test_range_pt(a) LOCAL;
CREATE INDEX
-- Query the OID of a partitioned index.
SELECT oid FROM pg_class WHERE relname = 'idx_range_a';
oid
-------
90250
(1 row)
```

- Query index partition information.

```
SELECT oid,relname,parttype,parentid,boundaries,indextblid FROM pg_partition WHERE parentid =
90250;
oid  | relname  | parttype | parentid | boundaries | indextblid
-------+----------+----------+----------+------------+------------
90255 | p5_a_idx | x        |    90250 |            |      49298
90254 | p4_a_idx | x        |    90250 |            |      49297
90253 | p3_a_idx | x        |    90250 |            |      49296
90252 | p2_a_idx | x        |    90250 |            |      49295
90251 | p1_a_idx | x        |    90250 |            |      49294
(5 rows)
```

Examples of Tool Functions

In non-multi-primary mode:

- pg_get_tabledef obtains the definition of a partitioned table. The input parameter can be the table OID or table name.

```
SELECT pg_get_tabledef('test_range_pt');

pg_get_tabledef
----------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
 SET search_path =
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
public;
                                        +
 CREATE TABLE test_range_pt
(
                                      +
    a
integer,
                                                 +
    b
integer,
                                                 +
    c
integer
                                                 +
 )
                                                    +
 WITH (orientation=row, compression=no, storage_type=USTORE,
segment=off)
          +
 PARTITION BY RANGE
(a)
                                         +

(
                                                  +
    PARTITION p1 VALUES LESS THAN (2000) TABLESPACE
pg_default,
              +
    PARTITION p2 VALUES LESS THAN (3000) TABLESPACE
pg_default,
              +
    PARTITION p3 VALUES LESS THAN (4000) TABLESPACE
pg_default,
              +
    PARTITION p4 VALUES LESS THAN (5000) TABLESPACE
pg_default,
              +
    PARTITION p5 VALUES LESS THAN (MAXVALUE) TABLESPACE
pg_default
            +
 )
                                                   +
 ENABLE ROW
MOVEMENT;
                                                 +
 CREATE INDEX idx_range_a ON test_range_pt USING ubtree (a) LOCAL(PARTITION p1_a_idx,
PARTITION p2_a_idx, PARTITION p3_a_idx, PARTITION p4_a_idx, PARTITION p5_a_idx)  WITH
(storage_type=USTORE) TABLESPACE pg_default;
(1 row)
```

- pg_stat_get_partition_tuples_hot_updated returns the number of tuples hot-updated in the partition with the specified partition ID.

  Insert 10 data records into and update the **p1** partition. Retrieve the number of tuples hot-updated in the **p1** partition.

```
INSERT INTO test_range_pt VALUES(generate_series(1,10),1,1);
INSERT 0 10
SELECT pg_stat_get_partition_tuples_hot_updated(49294);
pg_stat_get_partition_tuples_hot_updated
-----------------------------------------
0
(1 row)
UPDATE test_range_pt SET b = 2;
UPDATE 10
SELECT pg_stat_get_partition_tuples_hot_updated(49294);
pg_stat_get_partition_tuples_hot_updated
-----------------------------------------
10
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- pg_partition_size(oid,oid) returns the disk space occupied by the partition with the specified OID. The first **oid** indicates the table OID, and the second **oid** indicates the partition OID.

  Query the disk space of the **p1** partition.

  ```
  SELECT pg_partition_size(49290, 49294);
  pg_partition_size
  -------------------
  90112
  (1 row)
  ```

- pg_partition_size(text, text) returns the disk space occupied by the partition with the specified name. The first **text** indicates the table name, and the second **text** indicates the partition name.

  Query the disk space of the **p1** partition.

  ```
  SELECT pg_partition_size('test_range_pt', 'p1');
  pg_partition_size
  -------------------
  90112
  (1 row)
  ```

- pg_partition_indexes_size(oid,oid) returns the disk space occupied by the partitioned index with the specified OID. The first **oid** indicates the table OID, and the second **oid** indicates the partition OID.

  Query the disk space occupied by the index partition of the **p1** partition.

  ```
  SELECT pg_partition_indexes_size(49290, 49294);
  pg_partition_indexes_size
  ---------------------------
  204800
  (1 row)
  ```

- pg_partition_indexes_size(text,text) returns the disk space occupied by the partitioned index with the specified name. The first **text** indicates the table name, and the second **text** indicates the partition name.

  Query the disk space occupied by the index partition of the **p1** partition.

  ```
  SELECT pg_partition_indexes_size('test_range_pt', 'p1');
  pg_partition_indexes_size
  ---------------------------
  204800
  (1 row)
  ```

- pg_partition_filenode(partition_oid) obtains the filenode corresponding to the specified partitioned table OID.

  Query the filenode of the **p1** partition.

  ```
  SELECT pg_partition_filenode(49294);
  pg_partition_filenode
  -----------------------
  49294
  (1 row)
  ```

- pg_partition_filepath(partition_oid) returns the file path of the specified partition.

  Query the file path of the **p1** partition.

  ```
  SELECT pg_partition_filepath(49294);
  pg_partition_filepath
  -----------------------
  base/16521/49294
  (1 row)
  ```

- pg_partition_size(oid,oid) returns the disk space occupied by the partition with the specified OID. The first **oid** indicates the table OID, and the second **oid** indicates the partition OID.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Query the disk space of the **p1** partition.

```
SELECT pg_partition_size(49290, 49294);
pg_partition_size
------------------
90112
(1 row)
```

- pg_partition_size(text, text) returns the disk space occupied by the partition with the specified name. The first **text** indicates the table name, and the second **text** indicates the partition name.

  Query the disk space of the **p1** partition.

  ```
  SELECT pg_partition_size('test_range_pt', 'p1');
  pg_partition_size
  ------------------
  90112
  (1 row)
  ```

- pg_partition_indexes_size(oid,oid) returns the disk space occupied by the partitioned index with the specified OID. The first **oid** indicates the table OID, and the second **oid** indicates the partition OID.

  Query the disk space occupied by the index partition of the **p1** partition.

  ```
  SELECT pg_partition_indexes_size(49290, 49294);
  pg_partition_indexes_size
  --------------------------
  204800
  (1 row)
  ```

- pg_partition_indexes_size(text,text) returns the disk space occupied by the partitioned index with the specified name. The first **text** indicates the table name, and the second **text** indicates the partition name.

  Query the disk space occupied by the index partition of the **p1** partition.

  ```
  SELECT pg_partition_indexes_size('test_range_pt', 'p1');
  pg_partition_indexes_size
  --------------------------
  204800
  (1 row)
  ```

7.9 Storage Engine

7.9.1 Storage Engine Architecture

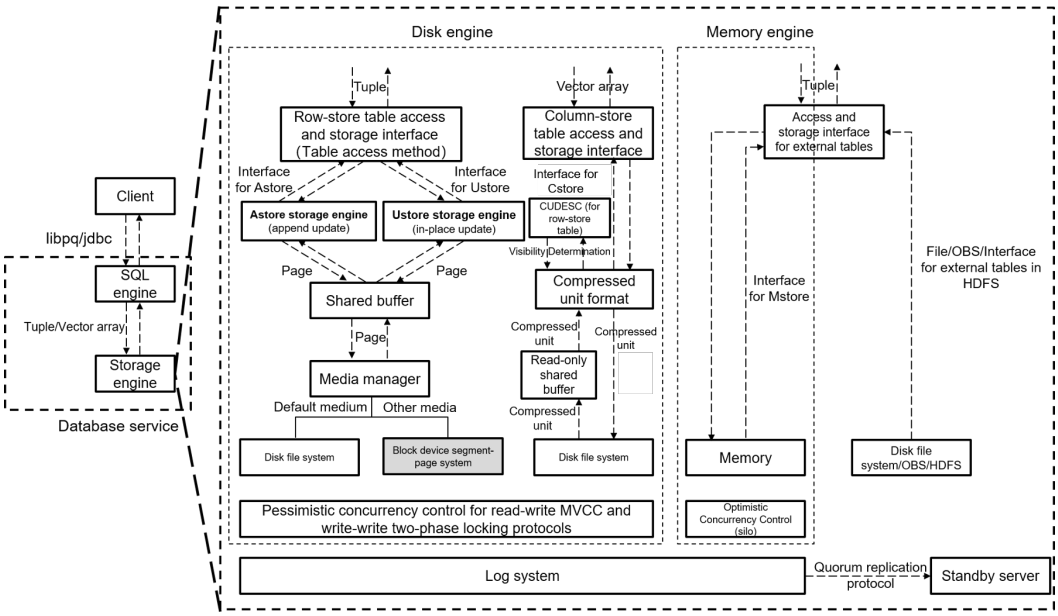
7.9.1.1 Overview

7.9.1.1.1 Static Compilation Architecture

From the perspective of the entire database service architecture, the storage engine upward connects to the SQL engine to provide or receive data in a standard format (tuple or vector array) for or from the SQL engine, and downward reads data from or writes data to storage media by a specific data organization mode such as page, compress unit, or other forms through specific APIs provided by the storage media. GaussDB enables database professionals to select dedicated storage engines for meeting specific application requirements through static compilation. To reduce interference to the execution engines, the row-store table access method (TableAM) layer is provided to shield the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

differences caused by the underlying row-store engines so that different row-store engines can evolve independently. See the following figure.



On this basis, the storage engines provide data persistence and reliability capabilities through the log system. The concurrency control (transaction) system ensures atomicity, consistency, and isolation between multiple read and write operations that are executed at the same time. The index system provides accelerated addressing and query capabilities for specific data. The primary/standby replication system provides high availability of the entire database service.

Row-store engines are oriented to online transaction processing (OLTP) scenarios, which are suitable for highly concurrent read and write operations on a small amount of data at a single point or within a small range. Row-store engines provide APIs upward to read tuples from or write tuples to the SQL engine, perform read and write operations downward on storage media by page through an extensible media manager, and improve read and write operation efficiency in the shared buffer by page. For concurrent read and write operations, multi-version concurrency control (MVCC) is used. For concurrent write and write operations, pessimistic concurrency control (PCC) based on the two-phase locking (2PL) protocol is used. Currently, the default media manager of row-store engines uses the disk file system API. Other types of storage media such as block devices will be supported in the future. The GaussDB row-store engine can be the append update-based Astore or in-place update-based Ustore.

7.9.1.1.2 Database Service Layer

From the technical perspective, a storage engine requires some infrastructure components.

Concurrency: The overhead of a storage engine can be reduced by properly employing locks, so as to improve overall performance. In addition, it provides functions such as multi-version concurrency control and snapshot reading.

Transaction: All transactions must meet the ACID requirements and their statuses can be queried.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Memory cache: Typically, storage engines cache indexes and data when accessing them. You can directly process common data in the cache pool, which facilitates the handling speed.

Checkpoint: Though storage engines are different, they all support incremental checkpoint/double write and full checkpoint/full page write. For different applications, you can select incremental checkpoint/double write or full checkpoint/full page write based on different conditions, which is transparent to storage engines.

Log: GaussDB uses physical logs. The write, transmission, and replay operations of physical logs are transparent to the storage engine.

7.9.1.2 Setting Up a Storage Engine

The storage engine has a great impact on the overall efficiency and performance of the database. Select a proper storage engine based on the actual requirements. You can run **WITH ([ORIENTATION | STORAGE_TYPE] [= value] [, ...])** to specify an optional storage parameter for a table or index. **Table 7-10** describes the parameters.

Table 7-10 Storage parameters

ORIENTATION	STORAGE_TYPE
ROW (default value): The data will be stored in rows.	[USTORE (default value)|ASTORE|Null]

If **ORIENTATION** is set to **ROW** and **STORAGE_TYPE** is left empty, the type of the created table is determined by the value of the **enable_default_ustore_table** parameter. (The parameter value can be **on** or **off**. The default value is **on**. For details about the parameter, see "Database GUC Parameters > GUC Parameters" in *Reference*.) If this parameter is set to **on**, a Ustore table is created. If this parameter is set to **off**, an Astore table is created.

Example:

```
gaussdb=# CREATE TABLE TEST(a int);
gaussdb=# \d+ test
                    Table "public.test"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

gaussdb=# CREATE TABLE TEST1(a int) with(orientation=row, storage_type=ustore);
gaussdb=# \d+ test1
Table "public.test1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, storage_type=ustore, compression=no, segment=off

gaussdb=# CREATE TABLE TEST2(a int) with(orientation=row, storage_type=astore);
gaussdb=# \d+ test2
Table "public.test2"
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, storage_type=astore, compression=no
gaussdb=# CREATE TABLE test4(a int) with(orientation=row);
gaussdb=# \d+
                                                List of relations
 Schema | Name  | Type |   Owner   |  Size   |                          Storage                          | Description
--------+-------+-------+-----------+---------+-----------------------------------------------------------
+-------------
 public | test  | table | z7ee88f3a | 0 bytes |
{orientation=row,compression=no,storage_type=USTORE,segment=off} |
 public | test1 | table | z7ee88f3a | 0 bytes |
{orientation=row,storage_type=ustore,compression=no,segment=off} |
 public | test2 | table | z7ee88f3a | 0 bytes | {orientation=row,storage_type=astore,compression=no}            |
 public | test4 | table | z7ee88f3a | 0 bytes |
{orientation=row,compression=no,storage_type=USTORE,segment=off} |
(5 rows)

gaussdb=# show enable_default_ustore_table;
 enable_default_ustore_table
-----------------------------
 on
(1 row)

gaussdb=#  DROP TABLE test;
gaussdb=#  DROP TABLE test1;
gaussdb=#  DROP TABLE test2;
gaussdb=#  DROP TABLE test4;
```

7.9.1.3 Storage Engine Update Description

7.9.1.3.1 GaussDB Kernel 505

- Ustore supports efficient storage of flexible fields.

- Ustore supports large-scale commercial use of TOAST.

- Added the page recovery and escape technologies to Ustore.

- Ustore supports the SMP technology.

7.9.1.3.2 GaussDB Kernel 503

- Adapted Ustore to distributed deployment, parallel query, global temporary tables, VACUUM FULL, and column constraints DEFERRABLE and INITIALLY DEFERRED.

- Added the online index rebuild function to Ustore.

- Enhanced B-tree empty page estimation for Ustore to improve the cost estimation accuracy of an optimizer.

- Added the storage engine reliability verification framework Diagnose Page/Page Verify to Ustore.

- Enhanced the view parsing, detection, and repair related to Ustore.

- Enhanced the WAL locating capability for Ustore. The gs_redo_upage system view is added to support constant replay of a single page and obtain and print any historical page, accelerating fault locating for damaged pages.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Extended the Ustore transaction directory's physical format for transaction slots for space reuse within a transaction.

- Added the online index creation function for Ustore.

- Adapted Ustore to the flashback function and ultimate RTO.

7.9.1.3.3 GaussDB Kernel R2

- Added Ustore for row store based on in-place update to implement separate storage of new and old data.

- Added rollback segments to Ustore.

- Added the synchronous, asynchronous, and in-page rollback to Ustore.

- Enhanced Ustore B-tree indexes for transactions.

- Added the flashback function to Astore to support table flashback, flashback query, flashback DROP, and flashback TRUNCATE.

- Ustore does not support the following features: distributed deployment, parallel query, table sampling, global temporary table, online creation, index rebuild, ultimate RTO, VACUUM FULL, and column constraints DEFERRABLE and INITIALLY DEFERRED.

7.9.2 Astore

7.9.2.1 Overview

The biggest difference between Astore and Ustore lies in whether the latest data and historical data are stored separately. Astore does not perform separated storage. Ustore only separates data, but does not separate indexes.

Astore Advantages

1. Astore does not have rollback segments, but Ustore does. For Ustore, rollback segments are very important. If rollback segments are damaged, data will be lost or even the database cannot be started. In addition, redo and undo operations are required for Ustore restoration. For Astore, because it does not have a rollback segment, old data is stored in the original files. Therefore, when the database crashes, complex restoration is not performed like that performed by a Ustore database.
2. Besides, the error "Snapshot Too Old" is not frequently reported, because old data is directly recorded in data files instead of rollback segments.
3. The rollback can be completed quickly because data is not deleted.

> **CAUTION**
>
> Rollback is complex. During transaction rollback, the modifications made by the transaction must be undone, the inserted records must be deleted, and the updated records must be rolled back. In addition, a large number of redo logs are generated during the rollback.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4. WAL in Astore is simpler than that in Ustore. Only data file changes need to be recorded in WALs. Rollback segment changes do not need to be recorded.
5. The recycle bin (flashback DROP and flashback TRUNCATE) function is supported.

7.9.3 Ustore

7.9.3.1 Overview

Ustore is an in-place update storage engine launched by GaussDB. The biggest difference between Ustore and Astore lies in that, the latest data and historical data (excluding indexes) are stored separately.

Ustore Advantages

- The latest data and historical data are stored separately. Compared with Astore, Ustore has a smaller scanning scope. The HOT chain of Astore is removed. Non-index columns, index columns, and heaps can be updated in-place without change to row IDs. Historical versions can be recycled in batches, and space bloat is controllable.
- The transaction information is added to B-tree indexes to perform MVCC independently. This increases the proportion of IndexOnlyScan and greatly reduces the number of times that TABLE ACCESS BY INDEX ROWID is executed.
- VACUUM is not the only way to clear historical data. Spaces are recycled independently. Indexes are decoupled from heap tables and can be cleared independently. The I/O stability is better.
- In the scenario where a large number of concurrent updates are performed on the same row, row ID offset may occur in an Astore table. The in-place update mechanism of Ustore ensures the stability of tuple row IDs and update latency.
- The flashback function is supported.

CAUTION

When modifying data pages, Ustore DML operations also trigger undo logs. Therefore, the update cost is higher. In addition, the scanning overhead of a single tuple is high because of replication (Astore returns pointers).

7.9.3.1.1 Ustore Features and Specifications

Restrictions

Category	Feature	Supported or Not
Transaction	Serializable	×

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Feature	Supported or Not
	DDL operations on a partitioned table in a transaction block	×
SQL	Table sampling/Materialized view/Key-value lock	×

Storage Specifications

1. The maximum number of columns in a data table is 1600.
2. Transaction directory (TD) **init_td** is a unique structure used by Ustore tables to store page transaction information. The number of TDs determines the maximum number of concurrent transactions supported on a page. When creating a table or index, you can specify the initial TD size **init_td**. The value range is [2,128], and the default value is **4**. A single page supports a maximum of 128 concurrent transactions.
3. The maximum tuple length of a Ustore table (excluding TOAST) cannot exceed 8192 – MAXALIGN(56 + init_td x 26 + 4), where **MAXALIGN** indicates 8-byte alignment. When the length of the inserted data exceeds the threshold, you will receive an error reporting that the tuple is too long to be inserted. The impact of **init_td** on the tuple length is as follows:
   - If the value of **init_td** is the minimum value **2**, the tuple length cannot exceed 8192 – MAXALIGN(56 + 2 x 26 + 4) = 8080 bytes.
   - If the value of **init_td** is the default value **4**, the tuple length cannot exceed 8192 – MAXALIGN(56 + 4 x 26 + 4) = 8024 bytes.
   - If the value of **init_td** is the maximum value **128**, the tuple length cannot exceed 8192 – MAXALIGN(56 + 128 x 26 + 4) = 4800 bytes.
4. The maximum number of index columns is 32. The maximum number of columns in a global partitioned index is 31.
5. The length of an index tuple cannot exceed (8192 – MAXALIGN(28 + 3 x 4 + 3 x 10) – MAXALIGN(42))/3, where **MAXALIGN** indicates 8-byte alignment. When the length of the inserted data exceeds the threshold, you will receive an error reporting that the tuple is too long to be inserted. As for the threshold, the index page header is 28 bytes, row pointer is 4 bytes, tuple CTID+INFO flag is 10 bytes, and page tail is 42 bytes.
6. The maximum rollback segment size is 16 TB.

7.9.3.1.2 Examples

Create a Ustore table.

Run the **CREATE TABLE** statement to create a Ustore table.

```
gaussdb=# CREATE TABLE ustore_table(a INT PRIMARY KEY, b CHAR (20)) WITH (STORAGE_TYPE=USTORE);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "ustore_table_pkey" for table
"ustore_table"
CREATE TABLE
gaussdb=# \d+ ustore_table
Table "public.ustore_table"
Column |     Type      | Modifiers | Storage  | Stats target | Description
--------+---------------+-----------+----------+--------------+-------------
```

```
a      | integer       | not null  | plain    |              |
b      | character(20) |           | extended |              |
Indexes:
"ustore_table_pkey" PRIMARY KEY, ubtree (a) WITH (storage_type=USTORE) TABLESPACE pg_default
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=ustore, compression=no, segment=off
```

Drop a Ustore table.

```
gaussdb=# DROP TABLE ustore_table;
 DROP TABLE
```

Create an index for a Ustore table.

Currently, Ustore supports only multi-version indexes of the B-tree type. In some scenarios, to distinguish from the B-tree indexes of Astore tables, the multi-version B-tree indexes of Ustore tables are also called UB-tree indexes. (For details about the UB-tree, see **UB-Tree**.) You can run the **CREATE INDEX** statement to create a UB-tree index for the "a" attribute of a Ustore table.

If no index type is specified for a Ustore table, a UB-tree index is created by default.

CAUTION

UB-tree indexes are classified into RCR UB-tree and PCR UB-tree. By default, an RCR UB-tree is created. If WITH option **index_txntype** is set to **pcr** or GUC parameter **index_txntype** is set to **pcr** during index creation, a PCR UB-tree is created.

```
gaussdb=# CREATE TABLE test(a int);
CREATE TABLE
gaussdb=# CREATE INDEX UB_tree_index ON test(a);
CREATE INDEX
gaussdb=# \d+ test
Table "public.test"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Indexes:
    "ub_tree_index" ubtree (a) WITH (storage_type=USTORE) TABLESPACE pg_default
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

-- Drop a Ustore table index.
gaussdb=# DROP TABLE test;
DROP TABLE
```

7.9.3.1.3 Best Practices of Ustore

How to Set init_td

Transaction directory (TD) is a unique structure used by Ustore tables to store page transaction information. The number of TDs determines the maximum number of concurrent transactions supported on a page. When creating a table or index, you can specify the initial TD size **init_td**, whose default value is **4**. That is,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

four concurrent transactions are supported to modify the page. The maximum value of **init_td** is **128**.

You can configure **init_td** based on the service concurrency requirements. You can also determine whether to adjust it based on the occurrence frequency of "wait available td" events during service running. Generally, **wait available td** is **0**. If the value of **wait available td** is not **0**, the "wait available td" event exists. In this case, you are advised to increase the value of **init_td** and observe the event for several times. If this case occasionally occurs, you are advised not to adjust the value because the extra TD slots occupy more space. You are advised to gradually increase the value in ascending order, such as 8, 16, 32, 48, ..., and 128, and check whether the number of wait events decreases significantly in this process. Use the minimum value of **init_td** with few wait events as the default value to save space. **wait available td** is one of the values of **wait_status**. **wait_status** indicates the waiting status of the current thread, including the waiting status details. You can query the value of **wait_status** in the PG_THREAD_WAIT_STATUS view. The value **none** indicates that the system is not waiting for any event. If there is a wait event, you can view the value of **wait available td**. The following is an example. For the configuration and detailed description of **init_td**, refer to "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Reference*. To view and modify the value of **init_td**, perform the following steps:

```
gaussdb=# CREATE TABLE test1(name varchar) WITH(storage_type = ustore, init_td=2);
gaussdb=# \d+ test1
                        Table "public.test1"
 Column |       Type        | Modifiers | Storage  | Stats target | Description
--------+-------------------+-----------+----------+--------------+-------------
 name   | character varying |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=ustore, init_td=2, compression=no, segment=off,
toast.storage_type=ustore, toast.toast_storage_type=enhanced_toast

gaussdb=# ALTER TABLE test1 SET(init_td=8);
gaussdb=# \d+ test1
                        Table "public.test1"
 Column |       Type        | Modifiers | Storage  | Stats target | Description
--------+-------------------+-----------+----------+--------------+-------------
 name   | character varying |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=ustore, compression=no, segment=off, init_td=8,
toast.storage_type=ustore, toast.toast_storage_type=enhanced_toast

gaussdb=# SELECT * FROM pg_thread_wait_status;
 node_name | db_name  |          thread_name          |    query_id     |       tid       |    sessionid    | lwtid |
psessionid | tlevel | smpid | wait_status | wait_event | locktag | lo
ckmode | block_sessionid | global_sessionid
-----------+----------+-------------------------------+-----------------+-----------------+-----------------+-------
+------------+--------+-------+-------------+------------+---------+---
-------+-----------------+------------------
 sgnode    |          | PageWriter                    |               0 | 139769678919424 | 139769678919424 | 16915
|          |     0 |     0 | none        | none       |         |
      |               | 0:0#0
 sgnode    |          | PageWriter                    |               0 | 139769736066816 | 139769736066816 | 16913
|          |     0 |     0 | none        | none       |         |
      |               | 0:0#0
 sgnode    |          | PageWriter                    |               0 | 139769707755264 | 139769707755264 | 16914
|          |     0 |     0 | none        | none       |         |
      |               | 0:0#0
 sgnode    |          | PageWriter                    |               0 | 139769761756928 | 139769761756928 | 16912
|          |     0 |     0 | none        | none       |         |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      |               | 0:0#0
 sgnode    |          | PageWriter                  |             0 | 139769783772928 | 139769783772928 | 16911
|           |     0 |     0 | none        | none       |        |
      |               | 0:0#0


gaussdb=# DROP TABLE test1;
DROP TABLE
```

How to Set fillfactor

fillfactor is a parameter used to describe the page filling rate and is directly related to the number and size of tuples that can be stored on a page and the physical space of a table. The default page filling rate of Ustore tables is 92%. The reserved 8% space is used for page update and TD list expansion. For the configuration and detailed description of **fillfactor**, refer to "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Reference*.

You can configure **fillfactor** after analyzing services. If only query or fixed-length update operations are performed after table data is imported, you can increase the page filling rate to 100%. If a large number of variable-length updates are performed after data is imported, you are advised to retain or decrease the page filling rate to reduce performance loss caused by cross-page update. To view and modify **fillfactor**, perform the following steps:

```
gaussdb=# CREATE TABLE test(a int) with(fillfactor=100);
gaussdb=# \d+ test
                    Table "public.test"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, fillfactor=100, compression=no, storage_type=USTORE, segment=off

gaussdb=# ALTER TABLE test set(fillfactor=92);
gaussdb=# \d+ test
                    Table "public.test"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off, fillfactor=92

gaussdb=# DROP TABLE test;
DROP TABLE
```

NOTICE

When you reset **init_td** for a table via ALTER TABLE, the new value takes effect only on pages inserted afterward. For existing pages, the number of TDs remains unchanged. To apply the new setting, you must execute VACUUM FULL.

Online Verification

Online verification, a unique feature for Ustore, effectively prevents logical page corruption caused by encoding errors during runtime. By default, the **ustore_attr** parameter enables verification for three modules (**UPAGE:UBTREE:UNDO**). Keep it enabled on the live network, except in performance-critical scenarios.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about the **ustore_attr** parameter, see "Database GUC Parameters > GUC Parameters > Developer Options" in *Reference*.

How to Set the Size of Rollback Segments

Generally, use the default size of rollback segments. To achieve optimal performance, you can adjust the parameters related to the rollback segment size in some scenarios. The specific scenarios and setting methods are as follows:

1. Historical data within a specified period needs to be retained.

   To use flashback or locate faults, you can change the value of **undo_retention_time** to retain more historical data. The default value of **undo_retention_time** is **0**. The value ranges from 0 to 3 days. The valid unit is second, minute, hour, or day. You are advised to set it to **900s**. Note that a larger value of **undo_retention_time** indicates more undo space usage and data space bloat, which further affects the data scan and update performance. When flashback is not used, you are advised to set **undo_retention_time** to a smaller value to reduce the disk space occupied by historical data and achieve optimal performance. You can use the following method to determine the new value that is more suitable for your service model:

   For the system function gs_stat_undo that collects undo statistics, if the input parameter is **false**, optimization suggestions on the **undo_space_limit_size**, **undo_limit_size_per_transaction**, and **undo_retention_time** parameters are provided. For details about the parameter values, see "SQL Reference > Functions and Operators > System Administration Functions > Undo System Functions" in *Reference*.

2. Historical data within a specified size needs to be retained.

   If long transactions or large transactions exist in your service, undo space may bloat. In this case, you need to increase the value of **undo_space_limit_size**. The default value of **undo_space_limit_size** is **256GB**, and the value ranges from 800 MB to 16 TB.

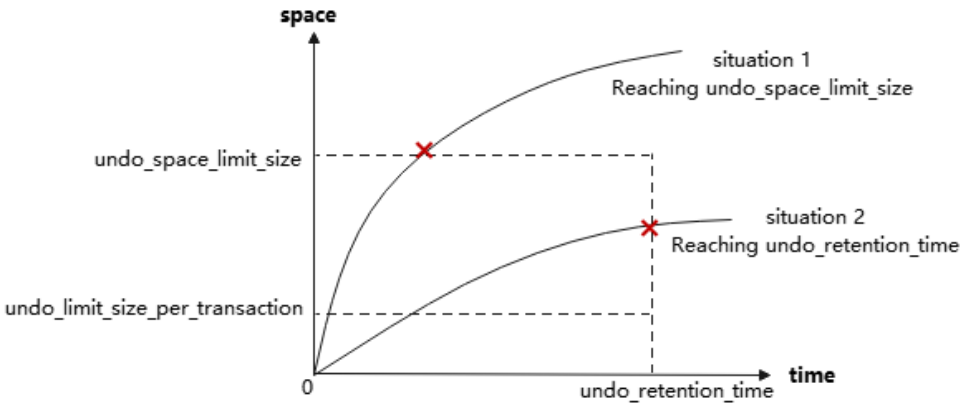
   If the disk space is sufficient, you are advised to double the value of **undo_space_limit_size**. In addition, a larger value of **undo_space_limit_size** indicates more disk space occupation and deteriorated performance. If no undo space bloat is found by querying **curr_used_undo_size** of the **gs_stat_undo()** view, you can restore the value to the original value.

   After adjusting the value of **undo_space_limit_size**, you can increase the value of **undo_limit_size_per_transaction**, which ranges from 2 MB to 16 TB. The default value is **32GB**. It is recommended that the value of **undo_limit_size_per_transaction** be less than or equal to that of **undo_space_limit_size**, that is, the threshold of the undo space allocated to a single transaction be less than or equal to the threshold of the total undo space.

3. The parameter adjustment priority is retained for historical data.

   Among **undo_retention_time**, **undo_space_limit_size**, and **undo_limit_size_per_transaction**, whichever limit is reached first will trigger the corresponding restriction.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.



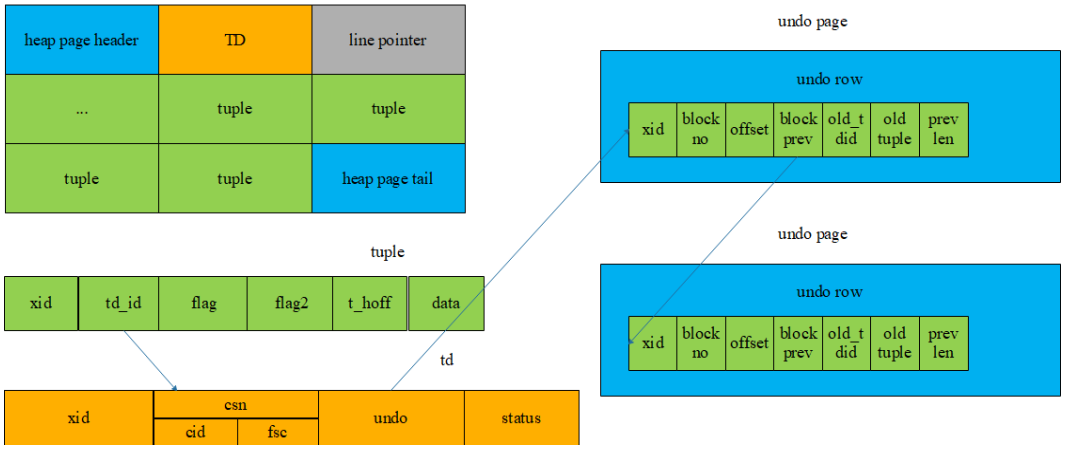
For example, the **undo_space_limit_size** parameter is set to **1 GB**, and the **undo_retention_time** parameter is set to **900s**. If the historical version data generated within 900s is less than 1 GB multiplied by 0.8, data recycling is performed every 900s. Otherwise, data recycling is performed when the volume of generated data reaches 1 GB multiplied by 0.8. In this case, if the disk space is sufficient, you can increase the value of **undo_space_limit_size**. If not, decrease the value of **undo_retention_time**.

7.9.3.2 Storage Format

7.9.3.2.1 RCR Uheap

RCR U-Heap Multi-Version Management

Ustore has made the following enhancements to the heap it uses, which is referred to as U-heap.



The multi-version management of Ustore row consistency read (RCR) is based on data row levels. Instead of commonly storing XIDs in data rows, Ustore records the XIDs in the transaction directory (TD) area of the page, saving page space. When a transaction modifies a record, historical data is recorded in the undo row. The generated undo row addresses (**zone_id**, **block no**, and **page offset**) are recorded in the TD slot to which **td_id** in tuple points, and new data is overwritten to the page. When a tuple is accessed, the tuple is restored along the version chain until the corresponding version is found.

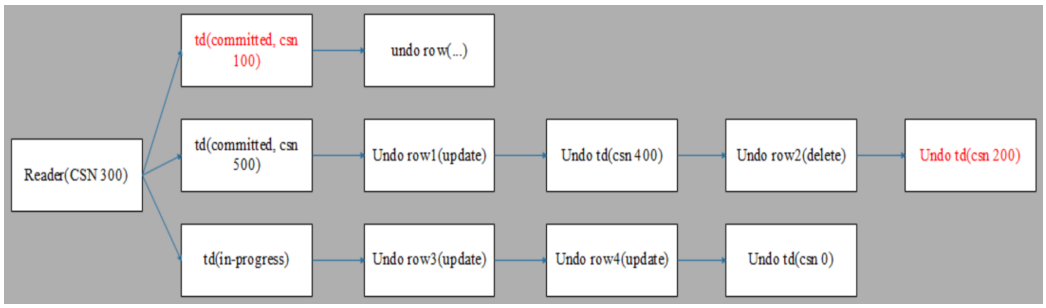
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

RCR U-Heap Visibility Mechanism

Ustore visibility is determined by building consistent versions of data rows. Old snapshots can be obtained from undo records. For example:

The figure below illustrates three query scenarios based on the MVCC visibility mechanism. Take a reader thread with a snapshot CSN of 300 as an example to query a visible data row version.

1. If the transaction to which the CSN points has been committed as recorded by the TD slot and the CSN recorded by the TD slot is less than the snapshot CSN, the transaction is visible to the snapshot. Furthermore, because the XID of the transaction that generated the current page version of this row is smaller than the XID recorded by the TD slot and the transaction has been committed, the latest page version is visible to the snapshot.
2. If the transaction to which the CSN points has been committed as recorded by the TD slot but the CSN recorded by the TD slot is greater than the snapshot CSN, the transaction is invisible to the snapshot. Therefore, the system must traverse the undo chain from newest to oldest to locate a visible version (that is, a version generated by a committed transaction with a CSN less than that of the snapshot). It ultimately finds the version with a CSN of 200.
3. If the transaction that reuses the TD slot is still in progress, making the transaction invisible to the snapshot. Again, the system traverses the undo chain from newest to oldest to find a visible version. The traversal concludes without finding a corresponding undo record, indicating that even the undo log generated by the transaction that created the current page version has been purged. This implies that all historical versions of this tuple have become visible to all possible snapshots. Therefore, the latest page version is the visible version.



RCR Uheap Free Space Management

Ustore uses the free space map (FSM) file to record the free space of each data page and organizes them in the tree structure. When you want to perform the INSERT operations or non-in-place UPDATE operations on a table, search an FSM file corresponding to the table to check whether the maximum free space in current FSM meets the requirement of the INSERT operation. If it does, perform the INSERT operation after the corresponding block number is returned. If not, expand the page.

The FSM structure corresponding to each table or partition is stored in an independent FSM file. The FSM file and the table data are stored in the same directory. For example, if the data file corresponding to table **t1** is **32181**, the corresponding FSM file is **32181_fsm**. FSM is stored in the format of data blocks, which are called FSM block. The logical structure among FSM blocks is a tree with

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

three layers of nodes. The nodes of the tree in logic are max heaps. Each time the FSM is searched from the root node to leaf nodes till an available page is returned for later service operations.

This structure may not keep real-time consistency with the actual available space of data pages and is maintained during DML execution. Ustore occasionally repairs and rebuilds FSM during autovacuum. When a user executes an INSERT DML statement, such as INSERT, NON-INPLACE UPDATE (new page), or MULTI INSERT, the FSM structure is queried to find a space where the current record can be inserted. After the DML operation is complete, the system determines whether to update the free space of the current page to the FSM based on the difference between the potential free space and the actual free space of the current page. The larger the difference, that is, the more the potential space is greater than the actual space, the higher the probability that the page is updated to the FSM. FSM records the potential free space of data pages. When a user searches a page to insert data, if the free space of the page is large, the data is directly inserted. Otherwise, if the potential space of the page is large, the page is cleared and then data is inserted. If the space is insufficient, search for the FSM structure again or expand the total number of pages. Updating the FSM structure involves DML statements, page cleaning, vacuum, page expansion, partition merging, and page scan.

7.9.3.2.2 UB-Tree

The B-tree is enhanced as follows, which is referred to as UB-tree.

- The transaction information is added to UB-tree indexes to perform MVCC independently. This increases the proportion of IndexOnlyScan and greatly reduces the number of times that TABLE ACCESS BY INDEX ROWID is executed.
- VACUUM is not the only way to clear historical data. Spaces are recycled independently. Indexes are decoupled from heap tables and can be cleared independently. The I/O stability is better.

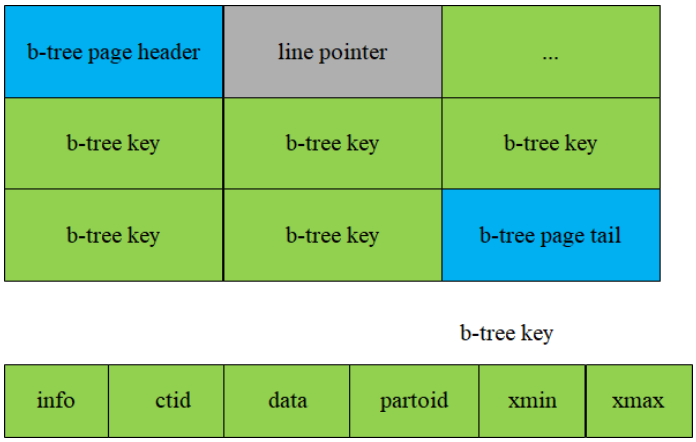
RCR UB-tree

RCR UB-Tree Multi-Version Management

The multi-version management of row consistency read (RCR) B-tree is based on data row levels. XIDs are recorded in data rows, resulting in larger key size and 5% to 20% index bloat. The latest and historical versions are on the B-tree, and the index does not record the undo information. Keys are inserted or deleted in the sequence of key+TID. Tuples with the same index column are sorted based on their TIDs as the second sorting keyword, and the values of **xmin** and **xmax** are added to the end of the key. During index splitting, multi-version information is migrated with key migration, as shown in **Figure 7-11**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

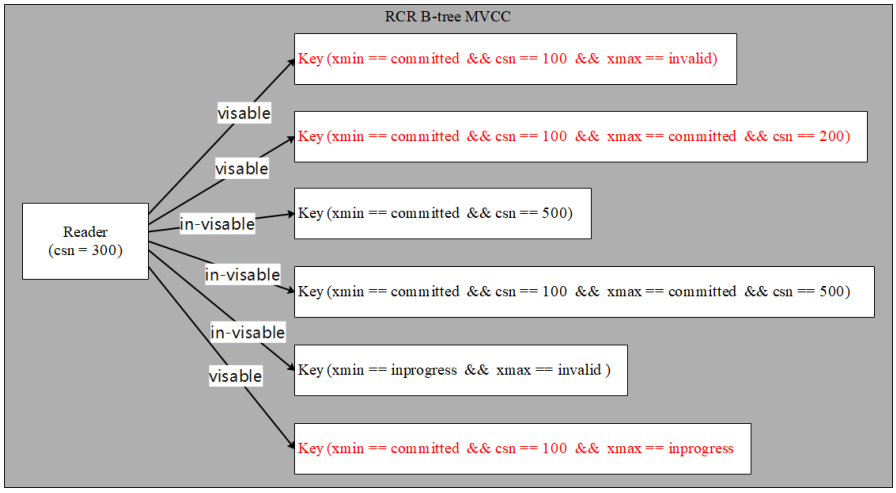
Figure 7-11 RCR UB-Tree Multi-Version Management



RCR UB-Tree Visibility Mechanism

The RCR UB-tree visibility is determined by the **xmin** and **xmax** on the key, which is similar to the **xmin** and **xmax** on the data row of the Astore heap table. The tuple visibility is determined based on the combination of the **xmin** and **xmax** commit status and CSN, as shown in **Figure 7-12**.

Figure 7-12 RCR UB-Tree Visibility Mechanism



Adding, Deleting, Modifying, and Querying RCR UB-Tree

- **Insert**: The insertion logic of UB-tree is basically not changed, except that you need to directly obtain the transaction information and fill in the **xmin** column during index insertion.
- **Delete**: The index deletion process is added to the UB-tree. The main procedure of index deletion is similar to that of index insertion (B-tree indexes do not maintain the version information so that the deletion operation is required). That is, obtain the transaction information, fill in the **xmax** column, and update **active_tuple_count** on pages. If the value of **active_tuple_count** is reduced to **0**, the system attempts to recycle the page.
- **Update**: For Ustore, data update operations on UB-tree index columns are different from those on Astore. Data updates include index column updates

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

and non-index column updates. **Figure 7-13** shows the process of UB-tree data updates.

Figure 7-13 UB-tree processing during data updates

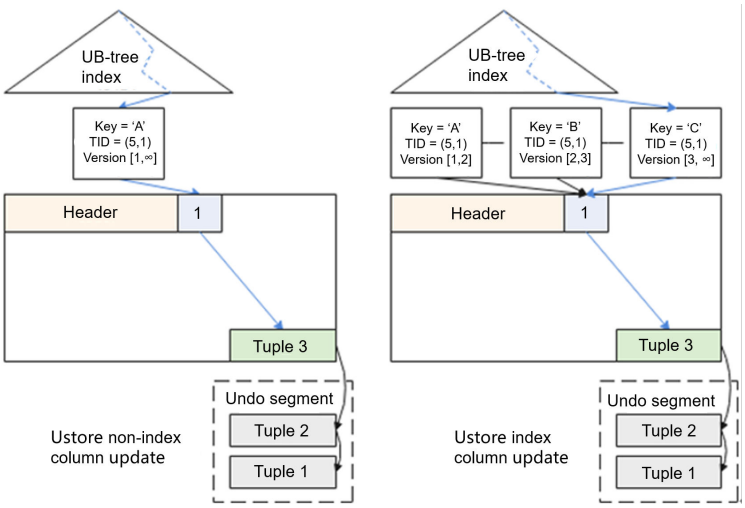


Figure 7-13 shows the differences between UB-tree updates on index columns and non-index columns.

- In the case of non-index column updates, the index does not change. The index tuple still points to the data tuple inserted for the first time. No data tuple is inserted into the U-heap. Instead, the U-heap modifies the current data tuple and saves historical data to the undo segment.
- When the index column is updated, a new index tuple is inserted into UB-tree and points to the same data linepointer and data tuple. To scan the historical data, you need to read it from the undo log.

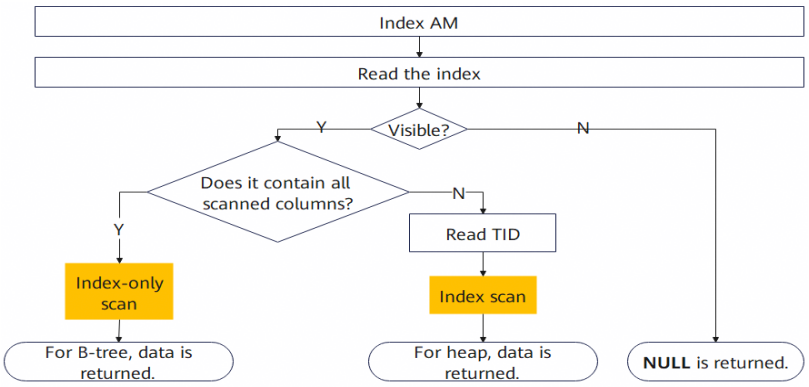
- **Scan**: When reading data, you can use index to speed up scanning. UB-tree supports multi-version management and visibility check of index data. The visibility check at the index layer improves the performance of index scan and index-only scan.

  For index scan:

  - If the index column contains all columns to be scanned (index-only scan), binary search is performed on indexes based on the scan conditions. If a tuple that meets the conditions is found, data is returned.
  - If the index columns do not cover all columns to be scanned (by index scan), binary search is performed on indexes based on the scan conditions to find TIDs of the tuples that meet the conditions, and then the corresponding data tuples are found in data tables based on the TIDs. For details, see **Figure 7-14**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

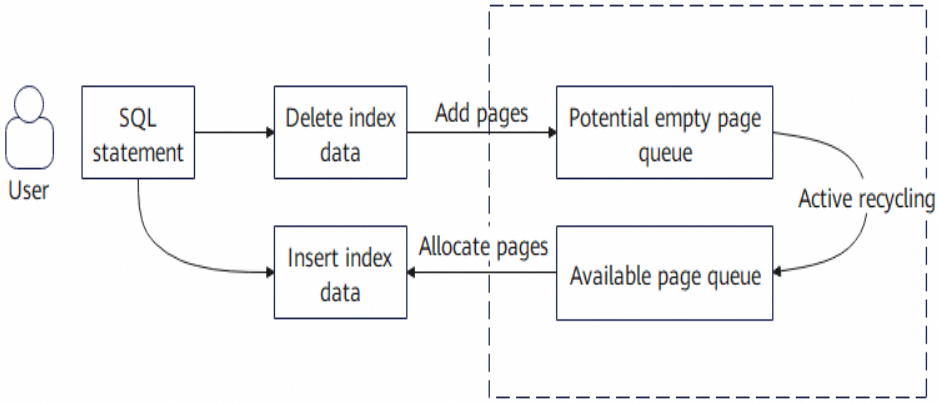
Figure 7-14 Corresponding data tuple



RCR UB-Tree Space Management

Currently, Astore indexes depend on autovacuum and FSM for space management. The space may not be recycled in a timely manner. However, Ustore indexes use the UB-tree recycle queue (URQ) to manage idle index space. The URQ contains two circular queues: potential empty page queue and available empty page queue. Completing space management of indexes in a DML process can effectively alleviate the sharp space bloat caused during the DML process. Index recycle queues are separately stored in FSM files corresponding to the B-tree indexes.

Figure 7-15 Index pages flow in the URQ



As shown in **Figure 7-15**, the index pages flow in the URQ as follows:

1. **From an empty page to a potential queue**

   The index page tail column records the number of active tuples (**activeTupleCount**) on the page. During the DML process, all tuples on a page are deleted, that is, when **activeTupleCount** is set to **0**, the index page is placed in the potential queue.

2. **From the potential queue to an available queue**

   The flow from a potential queue to an available queue mainly achieves an income and expense balance for the potential queue and ensure that pages are available for the available queue. That is, after an index empty page is used up in an available queue, at least one index page is transferred from a potential queue to the available queue. Besides, if a new index page is added

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

to a potential queue, at least one index page can be removed from the potential queue and inserted into the available queue.

3. **From the available queue to an empty page**

   When an empty index page is obtained during index splitting, the system first searches an available queue for an index page that can be reused. If such index page is found, it is directly reused. If no index page can be reused, physical page expansion is performed.

PCR UB-Tree

Compared with the RCR UB-tree, the PCR (page consistency read) UB-tree has the following features:

- The transaction information of the index tuple is managed by the TD slot.
- The undo operation is added. Before insertion and deletion, the undo log needs to be written. When a transaction is aborted, the rollback operation needs to be performed.
- Flashback is supported.

When creating an index, you can set WITH option **index_txntype** to **pcr** or set the GUC parameter **index_txntype** to **pcr** to create a PCR UB-tree. If the WITH option or GUC parameter is not specified, an RCR UB-tree is created by default. The PCR UB-tree does not support online creation, ultimate RTO replay, and standby-node read.

In the current version, it may take a long time to roll back a large number of PCR indexes. (The rollback time may increase exponentially as the data volume increases. If the data volume is too large, the rollback may fail to be completely executed.) The rollback time will be optimized in later versions. The following table describes the rollback time specifications in the current version.

Table 7-11 PCR index rollback time specifications

Type/Data Volume	100	1000	10,000	100,000	1 million
Rollback time with PCR indexes	0.692 ms	9.610 ms	544.678 ms	52,963.754 ms	89,440,029.048 ms
Rollback time without PCR indexes	0.226 ms	0.916 ms	8.974 ms	94.903 ms	1206.177 ms
Ratio	3.06	10.49	60.70	558.08	74,151.66

PCR UB-tree Multi-Version Management

Different from that of RCR UB-tree, page consistency read (PCR) multi-version management is based on pages. Transaction information of all tuples is managed by the TD slot.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PCR UB-tree Visibility Mechanism

PCR UB-tree visibility is determined by rolling back a page to a moment when the snapshot is visible to obtain a page on which all tuples are visible.

Adding, Deleting, Modifying, and Querying PCR UB-Tree

- **Insert**: The operation is basically the same as that of RCR UB-tree. The difference is that a TD needs to be allocated and undo logs need to be written before insertion.
- **Delete**: The operation is basically the same as that of RCR UB-tree. The difference is that a TD needs to be allocated and undo logs need to be written before deletion.
- **Update**:As with RCR UB-tree, this operation is converted into a Delete operation and an Insert operation.
- **Scan**: The operation is basically the same as that of RCR UB-tree. The difference is that the query operation needs to copy a CR page and roll back the CR page to the state visible to the snapshot. In this way, the tuples on the entire page are visible to the snapshot.

PCR UB-tree Space Management

The space management operations are basically the same as those of RCR UB-tree. The difference is that PCR UB-tree supports flashback. Therefore, the time point when the page can be recycled is changed from **OldestXmin** to **GlobalRecycleXid**.

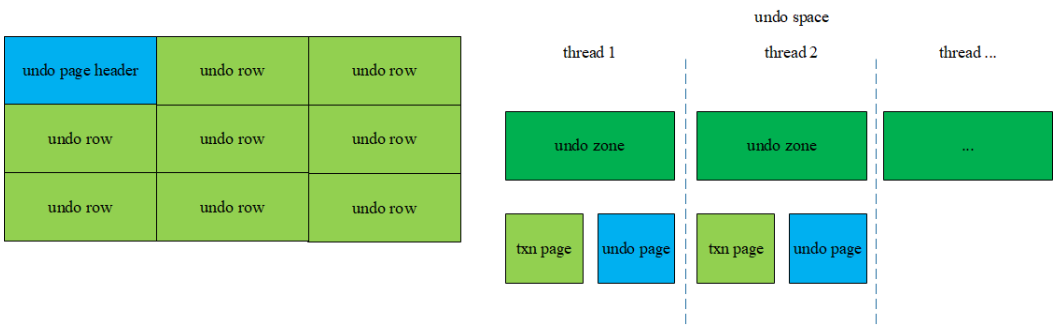
7.9.3.2.3 Undo

Historical data sets are stored in the *$node_dir*/**undo** directory, in which *$node_dir* indicates the database node path. The rollback segment log is a collection of all undo logs associated with a single write transaction. Permanent, unlogged, and temp tables are supported.

Rollback Segment Management

Figure 7-16 depicts rollback segment management.

Figure 7-16 Rollback segment management



1. In addition to managing some transaction pages (used to store metadata for transaction rollback), each undo zone also manages undo pages.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. Undo rows are stored on undo pages. Therefore, the modified data of historical versions is recorded on the undo pages.
3. Records on the undo pages are also data. Therefore, modifications on the undo pages are also recorded on the redo pages.

File Structure

To query whether the current rollback segment is stored in page or segment-page mode, query the system catalog. Currently, only the page mode is supported.

Example:

```
gaussdb=# SELECT * FROM gs_global_config where name like '%undostoragetype%';
      name       |  value
-----------------+---------
 undostoragetype | page
(1 row)
```

- When the rollback segment is stored by page:
  - Structure of the file where the txn page is stored
    ```
    $node_dir/undo/{permanent|unlogged|temp}/$undo_zone_id.meta.$segno
    ```
  - Structure of the file where the undo row is stored
    ```
    $node_dir/undo/{permanent|unlogged|temp}/$undo_zone_id.$segno
    ```

Space Management

The undo subsystem relies on the backend recycle thread to recycle free space. It recycles the space of the undo module on the primary node. As for the standby node, it recycles the space by replaying the Xlog. The recycle thread traverses the undo zones in use. The txn pages in the undo zone are scanned in the ascending order of XIDs. The transactions that have been committed or rolled back are also recycled. The commit time of transactions must be earlier than *$(current_time – undo_retention_time)*. For a transaction that needs to be rolled back during a traversal, the recycle thread adds an asynchronous rollback task for the transaction.

When the database has transactions that run for a long time and contain a large amount of modified data, or it takes a long time to enable flashback, the undo space may continuously expand. When the undo space is close to the value specified by **undo_space_limit_size**, forcible recycling is triggered. As long as a transaction has been committed or rolled back, the transaction may be recycled even if it is committed later than *$(current_time – undo_retention_time)*.

7.9.3.2.4 Enhanced TOAST

Overview

Enhanced TOAST is a technology used to process oversized fields. First, it reduces redundant information in the TOAST pointer so that more than 500 columns of oversized fields can be stored in a single table. Second, it optimizes the mapping between main tables and out-of-line storage tables so that the pg_toast_index table does not need to store such relationship, reducing user storage space. Lastly, the enhanced TOAST technology eliminates the dependency on OID allocation. Instead, it allows split data to be automatically linked, greatly improving write efficiency.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

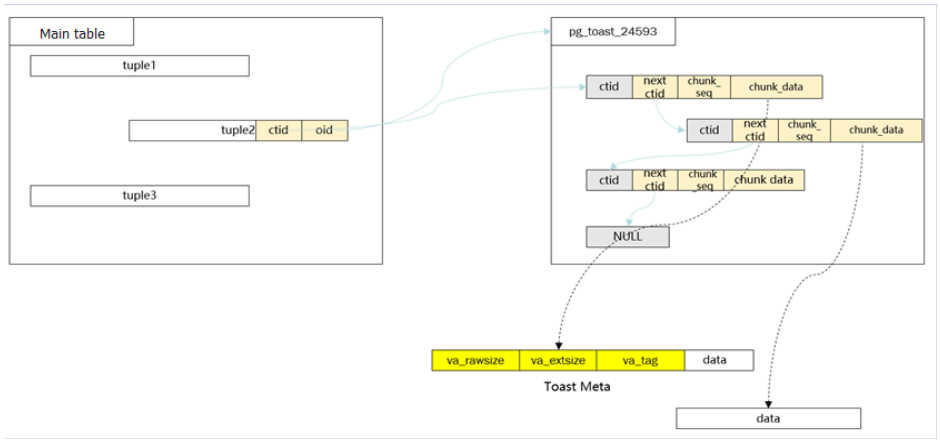
NOTE

- Astore does not support enhanced TOAST.
- No VACUUM FULL operation can be separately performed on out-of-line storage tables of the enhanced TOAST type.

Enhanced TOAST Storage Structure

The enhanced TOAST technology uses self-linking to handle dependencies between tuples. Out-of-line storage tables split overlong data into 2 KB linked list blocks. TOAST pointers in the main table point to corresponding data linked list headers in the out-of-line storage tables, as illustrated in **Figure 7-17**. This approach significantly simplifies the mapping between the main table and out-of-line storage tables, effectively enhancing data write and query performance.

Figure 7-17 Enhanced TOAST storage structure



Usage of Enhanced TOAST

The new GUC parameter **enable_enhance_toast_table** is used to control the out-of-line storage structure. For details about the **enable_enhance_toast_table** parameter, see "Database GUC Parameters > GUC Parameters > Enhanced TOAST" in *Reference*.

- **on**: The enhanced TOAST out-of-line storage table is used.
- **off**: The TOAST out-of-line storage table is used.

Adding, Deleting, Modifying, and Querying Enhanced TOAST

INSERT: The write conditions for triggering enhanced TOAST are the same as those for triggering TOAST. The INSERT logic remains unchanged except that the linking information is added when data is written.

DELETE: The DELETE process of enhanced TOAST does not depend on the TOAST data index. Instead, data is traversed and deleted based on the linking information between data.

UPDATE: The UPDATE process of enhanced TOAST is the same as that of TOAST.

DDL Operations Related to Enhanced TOAST

Create an enhanced TOAST table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Set the table storage type to enhanced TOAST or TOAST during table creation.

```
gaussdb=#  CREATE TABLE test_toast (id int, content text) with(toast.toast_storage_type=toast);
CREATE TABLE
gaussdb=#  \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=toast
gaussdb=#  DROP TABLE test_toast;
DROP TABLE
gaussdb=#  CREATE TABLE test_toast (id int, content text) with(toast.toast_storage_type=enhanced_toast);
CREATE TABLE
gaussdb=#  \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=enhanced_toast
gaussdb=#  DROP TABLE test_toast;
DROP TABLE
```

If the type of an out-of-line storage table is not specified during table creation, the table type depends on the **enable_enhance_toast_table** parameter.

```
-- Enable GUC parameters by following the instructions provided in section "Using Enhanced TOAST."
gaussdb=#  show enable_enhance_toast_table;
 enable_enhance_toast_table
----------------------------
 on
(1 row)
gaussdb=#  CREATE TABLE test_toast (id int, content text);
CREATE TABLE
gaussdb=#   \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=enhanced_toast

gaussdb=#   DROP TABLE test_toast;
DROP TABLE
gaussdb=#  SET enable_enhance_toast_table = off;
SET
gaussdb=#  show enable_enhance_toast_table;
 enable_enhance_toast_table
----------------------------
 off
(1 row)
gaussdb=#   CREATE TABLE test_toast (id int, content text);
CREATE TABLE
gaussdb=#  \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=toast
gaussdb=#  DROP TABLE test_toast;
DROP TABLE
```

Change the structure of an out-of-line storage table.

When the GUC parameter **enable_enhance_toast_table** is set to **on**, the structure of an out-of-line storage table can be changed to enhanced TOAST through the VACUUM FULL operation.

```
gaussdb=#   CREATE TABLE test_toast (id int, content text);
CREATE TABLE
gaussdb=#  \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=toast
gaussdb=#  VACUUM FULL test_toast;
VACUUM
gaussdb=#  \d+ test_toast
                    Table "public.test_toast"
 Column  |  Type   | Modifiers | Storage  | Stats target | Description
---------+---------+-----------+----------+--------------+-------------
 id      | integer |           | plain    |              |
 content | text    |           | extended |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off,
toast.storage_type=USTORE, toast.toast_storage_type=enhanced_toast
gaussdb=#  DROP TABLE test_toast;
DROP TABLE
```

Merge partitioned tables.

Different types of out-of-line storage table partitions can be merged.

NOTE

- For out-of-line storage partitions of the same type, the merge logic remains unchanged, that is, partitions are physically merged.
- For different types of out-of-line storage partitions, the out-of-line storage table generated after partitions are merged is an enhanced TOAST table and is only logically merged. The performance is inferior to that of physical merge.

```
gaussdb=#  CREATE TABLE test_partition_table(a int, b text)PARTITION BY range(a)(partition p1 values less
than (2000),partition p2 values less than (3000));
gaussdb=# SELECT relfilenode FROM pg_partition WHERE relname='p1';
 relfilenode
-------------
       17529
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=#  \d+ pg_toast.pg_toast_part_17529
TOAST table "pg_toast.pg_toast_part_17529"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Options: storage_type=ustore, toast_storage_type=toast

gaussdb=# SELECT relfilenode FROM pg_partition WHERE relname='p2';
 relfilenode
-------------
       17528
(1 row)

gaussdb=#  \d+ pg_toast.pg_toast_part_17528
TOAST table "pg_toast.pg_toast_part_17528"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_seq  | integer | plain
 next_chunk | tid     | plain
 chunk_data | bytea   | plain
Options: storage_type=ustore, toast_storage_type=enhanced_toast
gaussdb=#  ALTER TABLE test_partition_table MERGE PARTITIONS p1,p2 INTO partition p1_p2;
ALTER TABLE
gaussdb=#  SELECT reltoastrelid::regclass FROM pg_partition where relname='p1_p2';
         reltoastrelid
-------------------------------
 pg_toast.pg_toast_part_17559
(1 row)
gaussdb=#  \d+ pg_toast.pg_toast_part_17559
TOAST table "pg_toast.pg_toast_part_17559"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_seq  | integer | plain
 next_chunk | tid     | plain
 chunk_data | bytea   | plain
Options: storage_type=ustore, toast_storage_type=enhanced_toast
gaussdb=# DROP TABLE test_partition_table;
DROP TABLE
```

Enhanced TOAST O&M Management

Use gs_parse_page_bypath to parse the TOAST pointer information in the main table.

```
gaussdb=#  SELECT ctid,next_chunk,chunk_seq FROM pg_toast.pg_toast_part_17559;
 ctid  | next_chunk | chunk_seq
-------+------------+-----------
 (0,1) | (0,0)      |         1
 (0,2) | (0,1)      |         0
 (0,3) | (0,0)      |         1
 (0,4) | (0,3)      |         0
(4 rows)
gaussdb=#  SELECT gs_parse_page_bypath((SELECT * FROM
pg_relation_filepath('test_toast')),0,'uheap',false);
                      gs_parse_page_bypath
-----------------------------------------------------------------------
 ${data_dir}/gs_log/dump/1663_13113_17603_0.page
(1 row)
```

The parsing file **1663_13113_17603_0.page** stores the TOAST pointer information as follows:

```
Toast_Pointer:
   column_index: 1
      toast_relation_oid: 17608    -- OID of the out-of-line storage table
      ctid: (4, 1)                 -- Head pointer of the out-of-line storage data linked list
      bucket id: -1                -- Bucket ID information
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  column_index: 2
     toast_relation_oid: 17608
     ctid: (2, 1)
     bucket id: -1
```

Query enhanced TOAST data. You can determine the integrity of the link structure based on the queried enhanced TOAST table data.

```
gaussdb=#  SELECT ctid,next_chunk,chunk_seq FROM pg_toast.pg_toast_part_17559;
 ctid  | next_chunk | chunk_seq
-------+------------+-----------
 (0,1) | (0,0)      |         1
 (0,2) | (0,1)      |         0
 (0,3) | (0,0)      |         1
 (0,4) | (0,3)      |         0
(4 rows)
```

7.9.3.3 Ustore Transaction Model

GaussDB transaction basis:

1. An XID is not automatically allocated when a transaction is started, unless the first DML/DDL statement in the transaction is executed.
2. When a transaction ends, a commit log (CLOG) indicating the transaction commit state is generated. The states can be IN_PROGRESS, COMMITTED, ABORTED, or SUB_COMMITTED. Each transaction has two Clog status bits. Each byte on the Clog page indicates four transaction commit states.
3. When a transaction ends, a commit sequence number (CSN) is generated, which is an instance-level variable. Each XID corresponds to a unique CSN. The CSN can mark the following states of a transaction: committing, committed, rolled back, and frozen.

7.9.3.3.1 Transaction Commit

The commit policies for implicit and explicit transactions are as follows:

1. Implicit transaction. A single DML/DDL statement can automatically trigger an implicit transaction, which does not have explicit transaction block control statements (such as START TRANSACTION/BEGIN/COMMIT/END). After a DML/DDL statement ends, the transaction is automatically committed.
2. Explicit transaction. An explicit transaction uses an explicit statement, such as START TRANSACTION or BEGIN, to control the start of the transaction. The COMMIT and END statements control the commit of a transaction.

   A subtransaction must exist in an explicit transaction or stored procedure. The SAVEPOINT statement controls the start of a subtransaction, and the RELEASE SAVEPOINT statement controls the end of a subtransaction. If a subtransaction is not released during transaction committing, commit the subtransaction first. The transaction is not committed until all subtransactions are committed.

   Ustore supports READ COMMITTED. At the beginning of statement execution, the current system CSN is obtained for querying the current statement. The visible result of the entire statement is determined at the beginning of statement execution and is not affected by subsequent transaction modifications. By default, READ COMMITTED in the Ustore is consistent. Ustore also supports standard 2PC transactions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.9.3.3.2 Transaction Rollback

Rollback is a process in which a transaction cannot be executed if a fault occurs during transaction running. In this case, the system needs to cancel the modification operations that have been completed in the transaction. Astore and UB-tree do not have rollback segments. Therefore, there is no dedicated rollback operation. To ensure performance, the Ustore rollback process combines synchronous, asynchronous, and page-level rollbacks.

- **Synchronous rollback**

  Transaction rollback is triggered in any of the following scenarios:

  - The ROLLBACK keyword in a transaction block triggers a synchronous rollback.
  - If an error is reported during transaction running, the COMMIT keyword has the same function as ROLLBACK and triggers synchronous rollback.
  - If a fatal/panic error is reported during transaction running, the system attempts to synchronously roll back the transaction bound to the thread before the thread exits.

- **Asynchronous rollback**

  When the synchronous rollback fails or the system is restarted after breakdown, the undo recycling thread initiates an asynchronous rollback task for the transaction that is not rolled back completely and provides services for external systems immediately. The task initiation thread Undo Launch of asynchronous rollback starts the worker thread Undo Worker to execute the rollback task. The undo launch thread can start a maximum of five undo worker threads at the same time.

- **Page-level rollback**

  If a transaction has not been rolled back to the current page, but other transactions need to reuse the TD occupied by this transaction, the page-level rollback operation is performed for all modifications on the current page. Page-level rollback only rolls back modifications on the current page. Other pages are not involved.

  The rollback of a Ustore subtransaction is controlled by the ROLLBACK TO SAVEPOINT statement. After a subtransaction is rolled back, the transaction can continue to run. The rollback of a subtransaction does not affect the status of the transaction. If a subtransaction is not released during transaction rollback, roll back the subtransaction first. The transaction is not rolled back until all subtransactions are rolled back.

7.9.3.4 Flashback

Flashback is a part of the database recovery technology. It can be used to selectively cancel the impact of a committed transaction and restore data from incorrect manual operations. Before the flashback technology is used, the committed database modification can be retrieved only by means of restoring backup and PITR. The restoration takes several minutes or even hours. After the flashback technology is used, it takes only seconds to restore the DROP/TRUNCATE data committed in the database through FLASHBACK DROP and FLASHBACK TRUNCATE. In addition, the restoration time is irrelevant to the database size.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Astore supports only the flashback DROP/TRUNCATE function.
- Standby nodes do not support the flashback function.
- You can enable the flashback function as required. Note that enabling this function will cause performance deterioration.

7.9.3.4.1 Flashback Query

Context

Flashback query enables retrieving a table's snapshot data from a specific past point in time, making it possible to view and logically rebuild accidentally deleted or modified data. Flashback query is built on the MVCC mechanism to retrieve earlier data versions.

Prerequisites

The overall solution consists of three parts: earlier version retention, snapshot maintenance, and earlier version retrieval. For earlier version retention, the **undo_retention_time** parameter is introduced to specify the retention period for earlier versions. Data versions older than the specified period will be reclaimed and cleaned up. To use flashback query, contact an administrator to set this parameter to a value greater than 0.

Syntax

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
[TIMECAPSULE { TIMESTAMP | CSN } expression ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

In the syntax tree, **TIMECAPSULE {TIMESTAMP | CSN} expression** is a new expression for the flashback function. **TIMECAPSULE** indicates that the flashback function is used. **TIMESTAMP** and **CSN** indicate that the flashback function uses specific time point information or commit sequence number (CSN) information.

Parameters

- TIMESTAMP
  - Specifies a historical time point of the table data to be queried.
- CSN
  - Specifies a logical commit time point of the data in the entire database to be queried. Each CSN in the database represents a consistency point of the entire database. To query the data under a CSN means to query the data related to the consistency point in the database through SQL statements.

Note: When the time point is used for flashback, there may be a 3s error. To flash back to an operation point exactly, you need to use CSN for flashback. In GTM-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

free mode, there is no globally unified CSN. Therefore, flashback in CSN mode is not supported.

Examples

- Example (set **undo_retention_time** to a value greater than **0**):

```
gaussdb=# DROP TABLE IF EXISTS "public".flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
-- Create the flashtest table.
gaussdb=# CREATE TABLE "public".flashtest (col1 INT,col2 TEXT) with(storage_type=ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col1' as the distribution column by
default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Query the CSN.
gaussdb=# SELECT int8in(xidout(next_csn)) FROM gs_get_next_xid_csn();
  int8in
----------
 79351682
 79351682
 79351682
 79351682
 79351682
 79351682
(6 rows)
-- Query the current timestamp.
gaussdb=# SELECT now();
              now
-------------------------------
 2023-09-13 19:35:26.011986+08
(1 row)
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1,'INSERT1'),(2,'INSERT2'),(3,'INSERT3'),(4,'INSERT4'),
(5,'INSERT5'),(6,'INSERT6');
INSERT 0 6
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
------+---------
    3 | INSERT3
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
    6 | INSERT6
(6 rows)
-- Use flashback query to query the table at a CSN.
gaussdb=# SELECT * FROM flashtest TIMECAPSULE CSN 79351682;
 col1 | col2
------+------
(0 rows)
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
------+---------
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
    3 | INSERT3
    6 | INSERT6
(6 rows)
-- Use flashback query to query the table at a timestamp.
gaussdb=# SELECT * FROM flashtest TIMECAPSULE TIMESTAMP '2023-09-13 19:35:26.011986';
 col1 | col2
------+------
(0 rows)
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
------+---------
   1 | INSERT1
   2 | INSERT2
   4 | INSERT4
   5 | INSERT5
   3 | INSERT3
   6 | INSERT6
(6 rows)
-- Use flashback query to query the table at a timestamp.
gaussdb=# SELECT * FROM flashtest TIMECAPSULE TIMESTAMP to_timestamp ('2023-09-13
19:35:26.011986', 'YYYY-MM-DD HH24:MI:SS.FF');
 col1 | col2
------+------
(0 rows)
-- Use flashback query to query the table at a CSN and rename the table.
gaussdb=# SELECT * FROM flashtest AS ft TIMECAPSULE CSN 79351682;
 col1 | col2
------+------
(0 rows)
gaussdb=# DROP TABLE IF EXISTS "public".flashtest;
DROP TABLE
```

7.9.3.4.2 Flashback Table

Context

Flashback table enables you to restore a table to a specific point in time. When only one table or a group of tables are logically damaged instead of the entire database, this feature can be used to quickly restore the table data. Based on the MVCC mechanism, the flashback table deletes incremental data at a specified time point and after the specified time point and retrieves the data deleted at the specified time point and the current time point to restore table-level data.

Prerequisites

The overall solution consists of three parts: earlier version retention, snapshot maintenance, and earlier version retrieval. For earlier version retention, the **undo_retention_time** parameter is introduced to specify the retention period for earlier versions. Data versions older than the specified period will be reclaimed and cleaned up. To modify this parameter, contact an administrator.

Syntax

```
TIMECAPSULE TABLE table_name TO { TIMESTAMP | CSN } expression
```

Examples

```
gaussdb=# DROP TABLE IF EXISTS "public".flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
-- Create a table.
gaussdb=# CREATE TABLE "public".flashtest (col1 INT,col2 TEXT) with(storage_type=ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Query the CSN.
gaussdb=# SELECT int8in(xidout(next_csn)) FROM gs_get_next_xid_csn();
  int8in
----------
 79352065
 79352065
 79352065
 79352065
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 79352065
 79352065
(6 rows)
-- Query the current timestamp.
gaussdb=# SELECT now();
              now
-------------------------------
 2023-09-13 19:46:34.102863+08
(1 row)
-- View the flashtest table.
gaussdb=# SELECT * FROM flashtest;
 col1 | col2
------+------
(0 rows)
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1,'INSERT1'),(2,'INSERT2'),(3,'INSERT3'),(4,'INSERT4'),
(5,'INSERT5'),(6,'INSERT6');
INSERT 0 6
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
------+---------
    3 | INSERT3
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
    6 | INSERT6
(6 rows)
-- Flash a table back to a specific CSN.
gaussdb=# TIMECAPSULE TABLE flashtest TO CSN 79352065;
TimeCapsule Table
gaussdb=# SELECT * FROM flashtest;
 col1 | col2
------+------
(0 rows)
gaussdb=# SELECT now();
              now
-------------------------------
 2023-09-13 19:52:21.551028+08
(1 row)
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1,'INSERT1'),(2,'INSERT2'),(3,'INSERT3'),(4,'INSERT4'),
(5,'INSERT5'),(6,'INSERT6');
INSERT 0 6
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
------+---------
    3 | INSERT3
    6 | INSERT6
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
(6 rows)
-- Flash back the table to a specific timestamp before the current time.
gaussdb=# TIMECAPSULE TABLE flashtest TO TIMESTAMP to_timestamp ('2023-09-13 19:52:21.551028',
'YYYY-MM-DD HH24:MI:SS.FF');
TimeCapsule Table
gaussdb=# SELECT * FROM flashtest;
 col1 | col2
------+------
(0 rows)
gaussdb=# select now();
              now
-------------------------------
 2023-09-13 19:54:00.641506+08
(1 row)
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1,'INSERT1'),(2,'INSERT2'),(3,'INSERT3'),(4,'INSERT4'),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(5,'INSERT5'),(6,'INSERT6');
INSERT 0 6
gaussdb=# SELECT * FROM flashtest;
 col1 |  col2
------+---------
    3 | INSERT3
    6 | INSERT6
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
(6 rows)
-- Flash back the table to a specific timestamp after the current time.
gaussdb=# TIMECAPSULE TABLE flashtest TO TIMESTAMP '2023-09-13 20:54:00.641506';
ERROR:  The specified timestamp is invalid.
gaussdb=# SELECT * FROM flashtest;
 col1 | col2
------+------
    3 | INSERT3
    6 | INSERT6
    1 | INSERT1
    2 | INSERT2
    4 | INSERT4
    5 | INSERT5
(6 rows)
gaussdb=# DROP TABLE IF EXISTS "public".flashtest;
DROP TABLE
```

7.9.3.4.3 Flashback DROP/TRUNCATE

Context

- Flashback DROP enables you to restore tables that are dropped by mistake and their auxiliary structures, such as indexes and table constraints, from the recycle bin. Flashback DROP is based on the recycle bin mechanism. You can restore physical table files recorded in the recycle bin to restore dropped tables.
- Flashback TRUNCATE enables you to restore tables that are truncated by mistake and restore the physical data of the truncated tables and indexes from the recycle bin. Flashback TRUNCATE is based on the recycle bin mechanism. You can restore physical table files recorded in the recycle bin to restore truncated tables.

Prerequisites

- The GUC parameter **enable_recyclebin** is enabled (in the **gaussdb.conf** file). To modify it, contact an administrator.
- The **recyclebin_retention_time** parameter specifies the retention period for objects in the recycle bin. Objects older than the specified period will be automatically cleaned up. To modify this parameter, contact an administrator.

Syntax

- Drop a table.

  ```
  DROP TABLE table_name [PURGE]
  ```

- Purge objects in the recycle bin.

  ```
  PURGE { TABLE { table_name }
  | INDEX { index_name }
  | RECYCLEBIN
  }
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Flash back a dropped table.
  ```
  TIMECAPSULE TABLE { table_name } TO BEFORE DROP [RENAME TO new_tablename]
  ```
- Truncate a table.
  ```
  TRUNCATE TABLE { table_name } [ PURGE ]
  ```
- Flash back a truncated table.
  ```
  TIMECAPSULE TABLE { table_name } TO BEFORE TRUNCATE
  ```

Parameters

- **DROP/TRUNCATE TABLE table_name PURGE**

  Purges table data in the recycle bin by default.

- **PURGE RECYCLEBIN**

  Purges objects in the recycle bin.

- **TO BEFORE DROP**

  Retrieves dropped tables and their subobjects from the recycle bin.

  You can specify either the original user-defined name of the table or the system-generated name assigned to the object when it was dropped.

  – System-generated recycle bin object names are unique. Therefore, if you specify the system-generated name, the database retrieves that specified object. To see the content in your recycle bin, run **select * from gs_recyclebin;**.

  – If you specify a name and multiple objects in the recycle bin contain the name, the database retrieves the recently moved objects in the recycle bin. If you want to retrieve tables of earlier versions, do as follows:

    ▪ Specify the system-generated name of the table you want to retrieve.

    ▪ Run the **TIMECAPSULE TABLE... TO BEFORE DROP** statement until the table you want to retrieve is found.

  – When a dropped table is restored, only the base table name is restored, and the names of other objects remain the same as those in the recycle bin. You can run the DDL command to manually change the names of other objects as required.

  – The recycle bin does not support write operations such as DML, DCL, and DDL, and does not support DQL query operations (will be supported in later versions).

  – Between the flashback point and the current point, a statement has been executed to modify the table structure or to affect the physical structure. Therefore, the flashback fails. The error message "ERROR: The table definition of %s has been changed." is displayed when flashback is performed on a table where DDL operations have been performed. The error message "ERROR: recycle object %s desired does not exist" is displayed when flashback is performed on DDL operations, such as changing namespaces and table names.

  – When the **enable_recyclebin** parameter is enabled, if a table has a TRUNCATE trigger, TRUNCATE TABLE cannot activate the trigger.

- **RENAME TO**

  Specifies a new name for the table retrieved from the recycle bin.

- **TO BEFORE TRUNCATE**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Flashes back to the point in time before the TRUNCATE operation.

Examples

```
-- PURGE TABLE table_name; --
-- Check the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)

gaussdb=# DROP TABLE IF EXISTS flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)
-- Create the flashtest table.
gaussdb=# CREATE TABLE IF NOT EXISTS flashtest(id int, name text) with (storage_type = ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1, 'A');
INSERT 0 1
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)
-- Drop the flashtest table.
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
-- Check the recycle bin. The deleted table is moved to the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |    rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
-----------+---------+----------+------------------------------+----------------------+--------------+---------
+---------------+-------------------------------+--------------+-------------
--+--------------+----------+---------------+----------------+---------------+--------------+-------------+--------------
+----------------+-----------
    18591 |   12737 |    18585 | BIN$31C14EB4899$9737$0==$0   | flashtest            | d            |       0 |
79352606 | 2023-09-13 20:01:28.640664+08 |     79352595 |     7935259
5 |         2200 |       10 |             0 |          18585 | t            | t           | 225492       |         225492 |
    18591 |   12737 |    18588 | BIN$31C14EB489C$12D1BF60==$0 | pg_toast_18585       | d            |       2
|      79352606 | 2023-09-13 20:01:28.641018+08 |            0 |
0 |          99 |       10 |             0 |          18588 | f            | f           | 225492       |         225492 |
(2 rows)
-- Check the flashtest table. The table does not exist.
gaussdb=# SELECT * FROM flashtest;
ERROR:  relation "flashtest" does not exist
LINE 1: SELECT * FROM flashtest;
                      ^
-- Purge the table from the recycle bin.
gaussdb=# PURGE TABLE flashtest;
PURGE TABLE
-- Check the recycle bin. The table is deleted from the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)


-- PURGE INDEX index_name; --
gaussdb=# DROP TABLE IF EXISTS flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
-- Create the flashtest table.
gaussdb=# CREATE TABLE IF NOT EXISTS flashtest(id int, name text) WITH (storage_type = ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Create the flashtest_index index for the flashtest table.
gaussdb=# CREATE INDEX flashtest_index ON flashtest(id);
CREATE INDEX
-- View basic information about the flashtest table.
gaussdb=# \d+ flashtest
                    Table "public.flashtest"
 Column |  Type   | Modifiers | Storage  | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
 id     | integer |           | plain    |              |
 name   | text    |           | extended |              |
Indexes:
    "flashtest_index" ubtree (id) WITH (storage_type=USTORE) TABLESPACE pg_default
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=ustore, compression=no, segment=off,toast.storage_type=ustore,
toast.toast_storage_type=enhanced_toast

-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
-- Check the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |     rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
-----------+---------+----------+------------------------------+-----------------------+--------------+---------
+---------------+-------------------------------+--------------+------------
--+--------------+----------+---------------+----------------+---------------+-------------+--------------
+----------------+-----------
     18648 |   12737 |    18641 | BIN$31C14EB48D1$9A85$0==$0   | flashtest             | d            |       0 |
79354509 | 2023-09-13 20:40:11.360638+08 |     79354506 |     7935450
8 |         2200 |       10 |             0 |          18641 | t            | t           | 226642       |         226642 |
     18648 |   12737 |    18644 | BIN$31C14EB48D4$12E236A0==$0 | pg_toast_18641        | d            |       2
|     79354509 | 2023-09-13 20:40:11.36112+08  |             0 |
0 |          99 |       10 |             0 |          18644 | f            | f           | 226642       |         226642 |
     18648 |   12737 |    18647 | BIN$31C14EB48D7$9A85$0==$0   | flashtest_index       | d            |       1 |
79354509 | 2023-09-13 20:40:11.361246+08 |     79354508 |     7935450
8 |         2200 |       10 |             0 |          18647 | f            | t           | 0            |               0 |
(3 rows)

--Purge the flashtest_index index.
gaussdb=# PURGE INDEX flashtest_index;
PURGE INDEX
-- Check the recycle bin. The flashtest_index index is deleted from the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |     rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
```

```
-----------+---------+----------+-----------------------------+----------------------+--------------+---------
+---------------+-----------------------------+--------------+------------
--+--------------+----------+--------------+---------------+--------------+-------------+--------------
+---------------+-----------
    18648 |   12737 |    18641 | BIN$31C14EB48D1$9A85$0==$0   | flashtest            | d            |       0 |
79354509 | 2023-09-13 20:40:11.360638+08 |     79354506 |     7935450
8 |         2200 |       10 |            0 |         18641 | t            | t           | 226642       |        226642 |
    18648 |   12737 |    18644 | BIN$31C14EB48D4$12E236A0==$0 | pg_toast_18641       | d            |       2
|     79354509 | 2023-09-13 20:40:11.36112+08  |            0 |
0 |          99 |       10 |            0 |         18644 | f            | f           | 226642       |        226642 |
(2 rows)

-- PURGE RECYCLEBIN --
-- Purge the recycle bin.
gaussdb=# PURGE RECYCLEBIN;
PURGE RECYCLEBIN
-- Check the recycle bin. The recycle bin is cleared.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)

-- TIMECAPSULE TABLE { table_name } TO BEFORE DROP [RENAME TO new_tablename] --
gaussdb=# DROP TABLE IF EXISTS flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
-- Create the flashtest table.
gaussdb=# CREATE TABLE IF NOT EXISTS flashtest(id int, name text) with (storage_type = ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1, 'A');
INSERT 0 1
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
-- Check the recycle bin. The table is moved to the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |          rcyname          |    rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
-----------+---------+----------+-----------------------------+----------------------+--------------+---------
+---------------+-----------------------------+--------------+------------
--+--------------+----------+--------------+---------------+--------------+-------------+--------------
+---------------+-----------
    18658 |   12737 |    18652 | BIN$31C14EB48DC$9B2B$0==$0   | flashtest            | d            |       0 |
79354760 | 2023-09-13 20:47:57.075907+08 |     79354753 |     7935475
3 |         2200 |       10 |            0 |         18652 | t            | t           | 226824       |        226824 |
    18658 |   12737 |    18655 | BIN$31C14EB48DF$12E46400==$0 | pg_toast_18652       | d            |       2
|     79354760 | 2023-09-13 20:47:57.07621+08  |            0 |
0 |          99 |       10 |            0 |         18655 | f            | f           | 226824       |        226824 |
(2 rows)

-- Check the table. The table does not exist.
gaussdb=# SELECT * FROM flashtest;
ERROR:  relation "flashtest" does not exist
LINE 1: select * from flashtest;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                  ^
-- Flash back the dropped table.
gaussdb=# TIMECAPSULE TABLE flashtest to before drop;
TimeCapsule Table
-- Check the table. The table is restored to the state before the DROP operation.
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)

-- Check the recycle bin. The table is deleted from the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+--------------+--------------+---------+--------------+----------------
+--------------+-------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)

-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
gaussdb=# SELECT * FROM flashtest;
ERROR:  relation "flashtest" does not exist
LINE 1: SELECT * FROM flashtest;
                  ^
-- Check the recycle bin. The table is moved to the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |          rcyname           |       rcyoriginname         | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcy
changecsn | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge |
rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+-----------------------------+-----------------------------+--------------+---------
+---------------+-------------------------------+--------------+----
----------+--------------+----------+---------------+----------------+---------------+-------------+--------------
+----------------+-----------
    18664 |   12737 |    18652 | BIN$31C14EB48DC$9B4E$0==$0   | flashtest                   | d            |       0
|      79354845 | 2023-09-13 20:49:17.762977+08 |     79354753 |
 79354753 |         2200 |       10 |             0 |          18652 | t            | t           | 226824       |         226824 |
    18664 |   12737 |    18657 | BIN$31C14EB48E1$12E680A8==$0 | BIN$31C14EB48E1$12E45E00==$0 |
d            |       3 |      79354845 | 2023-09-13 20:49:17.763271+08 |     79354753 |
 79354753 |          99 |       10 |             0 |          18657 | f            | f           | 0            |              0 |
    18664 |   12737 |    18655 | BIN$31C14EB48DF$12E68698==$0 | BIN$31C14EB48DF$12E46400==$0 |
d            |       2 |      79354845 | 2023-09-13 20:49:17.763343+08 |            0 |
       0 |          99 |       10 |             0 |          18655 | f            | f           | 226824       |         226824 |
(3 rows)

-- Flash back the dropped table. The table name is rcyname in the recycle bin.
gaussdb=# TIMECAPSULE TABLE "BIN$31C14EB48DC$9B4E$0==$0" to before drop;
TimeCapsule Table
-- Check the recycle bin. The table is deleted from the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+--------------+--------------+---------+--------------+----------------
+--------------+-------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)

gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)

-- Drop the table.
```

```
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
-- Check the recycle bin. The table is moved to the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |       rcyoriginname         | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcy
changecsn | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge |
rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+------------------------------+------------------------------+--------------+---------
+---------------+------------------------------+--------------+----
----------+--------------+----------+---------------+----------------+---------------+-------------+--------------
+----------------+-----------
     18667 |   12737 |    18652 | BIN$31C14EB48DC$9B8D$0==$0   | flashtest                    | d            |       0
|      79354943 | 2023-09-13 20:52:14.525946+08 |      79354753 |
 79354753 |         2200 |       10 |             0 |          18652 | t            | t           | 226824       |         226824 |
     18667 |   12737 |    18657 | BIN$31C14EB48E1$1320B4F0==$0 | BIN$31C14EB48E1$12E680A8==$0 |
d            |       3 |      79354943 | 2023-09-13 20:52:14.526319+08 |     79354753 |
 79354753 |          99 |       10 |             0 |          18657 | f            | f           | 0            |              0 |
     18667 |   12737 |    18655 | BIN$31C14EB48DF$1320BAE0==$0 | BIN$31C14EB48DF$12E68698==$0 |
d            |       2 |      79354943 | 2023-09-13 20:52:14.526423+08 |            0 |
       0 |          99 |       10 |             0 |          18655 | f            | f           | 226824       |         226824 |
(3 rows)

-- Check the table. The table does not exist.
gaussdb=# SELECT * FROM flashtest;
ERROR:  relation "flashtest" does not exist
LINE 1: SELECT * FROM flashtest;
                      ^
-- Flash back the dropped table and rename the table.
gaussdb=# TIMECAPSULE TABLE flashtest to before drop rename to flashtest_rename;
TimeCapsule Table
-- Check the original table. The table does not exist.
gaussdb=# SELECT * FROM flashtest;
ERROR:  relation "flashtest" does not exist
LINE 1: SELECT * FROM flashtest;
                      ^
-- Check the renamed table. The table exists.
gaussdb=# SELECT * FROM flashtest_rename;
 id | name
----+------
  1 | A
(1 row)

-- Check the recycle bin. The table is deleted from the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)
-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS flashtest_rename;
DROP TABLE
-- Clear the recycle bin.
gaussdb=# PURGE RECYCLEBIN;
PURGE RECYCLEBIN
-- Check the recycle bin. The recycle bin is cleared.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- TIMECAPSULE TABLE { table_name } TO BEFORE TRUNCATE --
gaussdb=# DROP TABLE IF EXISTS flashtest;
NOTICE:  table "flashtest" does not exist, skipping
DROP TABLE
-- Create the flashtest table.
gaussdb=# CREATE TABLE IF NOT EXISTS flashtest(id int, name text) WITH (storage_type = ustore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Insert data.
gaussdb=# INSERT INTO flashtest VALUES(1, 'A');
INSERT 0 1
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)

-- Truncate a table.
gaussdb=# TRUNCATE TABLE flashtest;
TRUNCATE TABLE
-- Check the recycle bin. The table data is moved to the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |    rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
-----------+---------+----------+------------------------------+----------------------+--------------+---------
+---------------+-------------------------------+--------------+------------
--+--------------+----------+---------------+----------------+---------------+-------------+--------------
+----------------+-----------
    18703 |   12737 |    18697 | BIN$31C14EB4909$9E4C$0==$0   | flashtest            | t            |       0 |
79356608 | 2023-09-13 21:24:42.819863+08 |     79356606 |     7935660
6 |         2200 |       10 |             0 |          18697 | t            | t           | 227927       |        227927 |
    18703 |   12737 |    18700 | BIN$31C14EB490C$132FE3F0==$0 | pg_toast_18697       | t            |       2
|     79356608 | 2023-09-13 21:24:42.820358+08 |            0 |
0 |          99 |       10 |             0 |          18700 | f            | f           | 227927       |        227927 |
(2 rows)

-- Check the table. The table is empty.
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
(0 rows)

-- Flash back a truncated table.
gaussdb=# TIMECAPSULE TABLE flashtest to before truncate;
TimeCapsule Table
-- Check the table. The data in the table is restored.
gaussdb=# SELECT * FROM flashtest;
 id | name
----+------
  1 | A
(1 row)

-- Check the recycle bin.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid |           rcyname            |    rcyoriginname     | rcyoperation | rcytype |
rcyrecyclecsn |        rcyrecycletime         | rcycreatecsn | rcychangecs
n | rcynamespace | rcyowner | rcytablespace | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid |
rcyfrozenxid64 | rcybucket
-----------+---------+----------+------------------------------+----------------------+--------------+---------
+---------------+-------------------------------+--------------+------------
--+--------------+----------+---------------+----------------+---------------+-------------+--------------
+----------------+-----------
    18703 |   12737 |    18700 | BIN$31C14EB490C$13300228==$0 | pg_toast_18697       | t            |       2
|     79356610 | 2023-09-13 21:24:42.872732+08 |            0 |
0 |          99 |       10 |             0 |          18706 | f            | f           | 0            |        227928 |
    18703 |   12737 |    18697 | BIN$31C14EB4909$9E4D$0==$0   | flashtest            | t            |       0 |
```

```
79356610 | 2023-09-13 21:24:42.872792+08 |     79356606 |     7935660
6 |         2200 |       10 |             0 |          18704 | t             | t           | 0           |        227928 |
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE IF EXISTS flashtest;
DROP TABLE
-- Clear the recycle bin.
gaussdb=# PURGE RECYCLEBIN;
PURGE RECYCLEBIN
-- Check the recycle bin. The recycle bin is cleared.
gaussdb=# SELECT * FROM gs_recyclebin;
 rcybaseid | rcydbid | rcyrelid | rcyname | rcyoriginname | rcyoperation | rcytype | rcyrecyclecsn |
rcyrecycletime | rcycreatecsn | rcychangecsn | rcynamespace | rcyowner | rcytablespace
 | rcyrelfilenode | rcycanrestore | rcycanpurge | rcyfrozenxid | rcyfrozenxid64 | rcybucket
-----------+---------+----------+---------+---------------+--------------+---------+---------------+----------------
+--------------+--------------+--------------+----------+--------------
-+----------------+---------------+-------------+--------------+----------------+-----------
(0 rows)
```

7.9.3.5 Common View Tools

View Type	Type	Description	Application Scenario	Function
Parsing	All types	Parses a specified table page and returns the path for storing the parsed content.	• Page information viewing • Tuple (non-user data) information • Damaged pages and tuples • Tuple visibility problems • Verification errors	gs_parse_page_bypath
	Index recycle queue (URQ)	Parses key information in the URQ.	• UB-tree index space expansion • UB-tree index space recycle exceptions • Verification errors	gs_urq_dump_stat

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Type	Type	Description	Application Scenario	Function
	Rollback segment (undo)	Parses the specified undo record, excluding old tuple data.	• Expanded undo space • Undo recycling exceptions • Rollback exceptions • Routine maintenance • Verification errors • Visibility judgment exceptions • Parameter modifications	gs_undo_dump_record
		Parses all undo records generated by a specified transaction, excluding old tuple data.		gs_undo_dump_xid
		Parses all information about transaction slots in a specified undo zone.		gs_undo_translot_dump_slot
		Parses the transaction slot information of a specified transaction, including the XID and the range of undo records generated by the transaction.		gs_undo_translot_dump_xid
		Parses the metadata of a specified undo zone and displays the pointer usage of undo records and transaction slots.		gs_undo_meta_dump_zone
		Parses the undo space metadata corresponding to a specified undo zone and displays the file usage of undo records.		gs_undo_meta_dump_spaces
		Parses the slot space metadata corresponding to a specified undo zone and displays the file usage of transaction slots.		gs_undo_meta_dump_slot
		Parses the data page and all data of historical versions and returns the path for storing the parsed content.		gs_undo_dump_parsepage_mv
	Write ahead log (WAL)	Parses Xlog within the specified LSN range and returns the path for storing parsed content. You can use pg_current_xlog_location() to obtain the current Xlog position.	• WAL errors • Log replay errors • Damaged pages	gs_xlogdump_lsn
		Parses Xlog of a specified XID and returns the path for storing parsed content. You can use txid_current() to obtain the current XID.		gs_xlogdump_xid
		Parses logs corresponding to a specified table page and returns the path for storing the parsed content.		gs_xlogdump_tablepath

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Type	Type	Description	Application Scenario	Function
		Parses the specified table page and logs corresponding to the table page and returns the path for storing the parsed content. It can be regarded as one execution of gs_parse_page_bypath and gs_xlogdump_tablepath. The prerequisite for executing this function is that the table file exists. To view logs of deleted tables, call gs_xlogdump_tablepath.		gs_xlogdump_parsepage_tablepath
Collecting	Rollback segment (undo)	Displays the statistics of the Undo module, including the usage of undo zones and undo links, creation and deletion of undo module files, and recommended values of undo module parameters.	• Undo space expansion • Undo resource monitoring	gs_stat_undo
	Write ahead log (WAL)	Collects statistics of the memory status table when WALs are written to disks.	• WAL write/disk flushing monitoring • Suspended WAL write/disk flushing	gs_stat_wal_entrytable
		Collects WAL statistics about the disk flushing status and location.		gs_walwriter_flush_position
		Collects WAL statistic about the frequency of disk flushing, data volume, and flushing files.		gs_walwriter_flush_stat
Validation	Heap table/ Index	Checks whether the disk page data of tables or index files is normal offline.	• Damaged pages and tuples • Visibility issues • Log replay errors	ANALYZE VERIFY
		Checks whether physical files of the current database in the current instance are lost.	Lost files	gs_verify_data_file
	Index recycle (URQ)	Checks whether the data of the URQ (potential queue/available queue/single page) is normal.	• UB-tree index space expansion • UB-tree index space reclamation exceptions	gs_verify_urq

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Type	Type	Description	Application Scenario	Function
	Rollback segment (undo)	Checks whether undo records are normal offline.	• Abnormal or damaged undo records • Visibility issues • Abnormal or damaged rollback	gs_verify_undo_record
		Checks whether the transaction slot data is normal offline.	• Abnormal or damaged undo records • Visibility issues • Abnormal or damaged rollback	gs_verify_undo_slot
		Checks whether the undo metadata is normal offline.	• Node startup failure caused by undo metadata • Undo space reclamation exceptions • Outdated snapshots	gs_verify_undo_meta
Restoration	Heap table/ Index/ Undo file	Restores lost physical files on the primary node based on the standby node.	Lost heap tables/ Indexes/undo files	gs_repair_file
	Heap table/ Index/ Undo page	Checks and restores damaged pages on the primary node based on the standby node.	Damaged heap tables/ indexes/undo pages	gs_verify_and_tryrepair_page

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Type	Type	Description	Application Scenario	Function
		Restores the pages of the primary node based on the pages of the standby node.		gs_repair_page
		Modifies the bytes of the page backup based on the offset.		gs_edit_page_bypath
		Overwrites the modified page to the target page.		gs_repair_page_bypath
	Rollback segment (undo)	Rebuilds undo metadata. If the undo metadata is proper, rebuilding is not required.	Abnormal or damaged undo metadata	gs_repair_undo_byzone
	Index recycle queue (URQ)	Rebuilds the URQ.	Abnormal or damaged URQ	gs_repair_urq

7.9.3.6 Common Problems and Troubleshooting Methods

7.9.3.6.1 Snapshot Too Old

Undo space cannot save historical data if the execution time of the query SQL statement is too long or other reasons. Therefore, an error may be reported if the historical data is forcibly recycled. Generally, the rollback segment space needs to be expanded. However, the specific problem needs to be analyzed.

Undo Space Recycling Blocked by Long Transactions

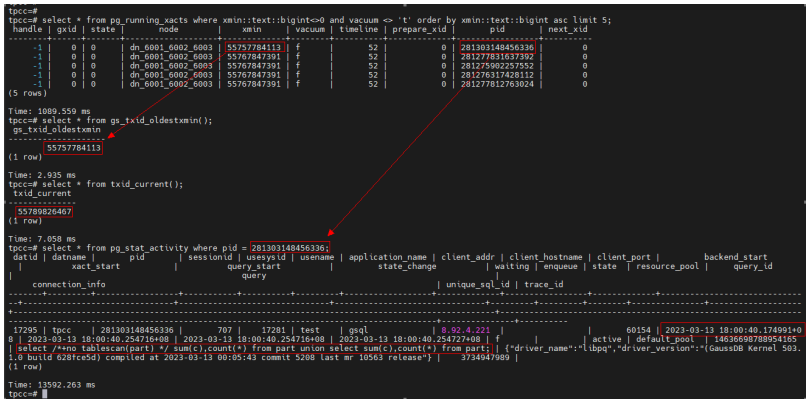
Symptom

1. The following error information is printed in gs_log:
   ```
   snapshot too old! the undo record has been forcibly discarded
   xid xxx, the undo size xxx of the transaction exceeds the threshold xxx. trans_undo_threshold_size xxx,undo_space_limit_size xxx.
   ```
   In the actual error information, *xxx* indicates the actual data.
2. The value of **global_recycle_xid** (global recycling XID of the Undo subsystem) does not change for a long time.
   ```
   gaussdb=# select * from gs_undo_meta_dump_slot(1,-1);
    zone_id | allocate | recycle | frozen_xid | global_frozen_xid | recycle_xid | global_recycle_xid
   ---------+----------+---------+------------+-------------------+-------------+--------------------
          1 | 280      | 248     | 17028      | 17028             | 17025       | 17028
   (1 row)
   ```
3. Long transactions exist in the pg_running_xacts and pg_stat_activity views, blocking the progress of **oldestxmin** and **global_recycle_xid**. If the value of **xmin** obtained by querying active transactions in pg_running_xacts is the same as that of gs_txid_oldestxmin and the thread execution time obtained

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

by querying pg_stat_activity based on a PID is too long, the long transactions are suspended and recycled.

```
SELECT * FROM pg_running_xacts where xmin::text::bigint<>0 and vacuum <> 't' order by xmin::text::bigint asc limit 5;
SELECT * FROM gs_txid_oldestxmin();
SELECT * FROM pg_stat_activity WHERE pid = Thread PID where the long transaction exists
```



Solution

Use **pg_terminate_session(pid, sessionid)** to terminate the sessions of the long transactions. (Note: There is no fixed quick restoration method for long transactions. Forcibly ending the execution of SQL statements is a common but high-risk operation. Exercise caution when performing this operation. Before performing this operation, please confirm with the administrator and Huawei technical personnel to prevent service failures or errors.)

Slow Undo Space Recycling Caused by Many Rollback Transactions

Symptom

The gs_async_rollback_xact_status view shows that there are a large number of transactions to be rolled back, and the number of transactions to be rolled back remains unchanged or keeps increasing.

```
SELECT * FROM gs_async_rollback_xact_status();
```

Solution

Increase the number of asynchronous rollback threads by increasing the value of **max_undo_workers**. For details about the **max_undo_workers** parameter, see "Database GUC Parameters > GUC Parameters > Rollback Parameters" in *Reference*.

7.9.3.6.2 Storage Test Error

During service execution, if a data page, index, or undo page changes, logic damage detection is performed before the page is locked. If a page damage is detected, logs containing the keyword "storage test error" are exported to the database running log file **gs_log**. The page is restored to the status before the modification after rollback.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Symptom

The keyword "storage test error" is printed in **gs_log**.

Solution

Contact Huawei technical support.

7.9.3.6.3 Read-on-Standby Error: "Index scan conflicts with replay. Try again later. The original maxoff is <number>, but now it is <number>."

Symptom

A read-on-standby error (error code 43244) is reported: "Index scan conflicts with replay. Try again later."

Analysis

When parallel or serial replay is enabled (if the GUC parameters **recovery_parse_workers** and **recovery_max_workers** are both set to **1**, serial replay is enabled; if **recovery_parse_workers** is set to **1** and **recovery_max_workers** is greater than 1, parallel replay is enabled): If the query thread of the standby node scans indexes, a read lock is added to the index page. Each time a tuple is scanned, the visibility is checked. If the transaction corresponding to the tuple is in the committing state, the visibility is checked after the transaction is committed. Transactions committed on the standby node depend on the log replay thread. During this process, the index page is modified. Therefore, a lock is required. The query thread releases the lock of the index page during waiting. Otherwise, the query thread waits for the replay thread to commit the transaction, and the replay thread waits for the query thread to release the lock.

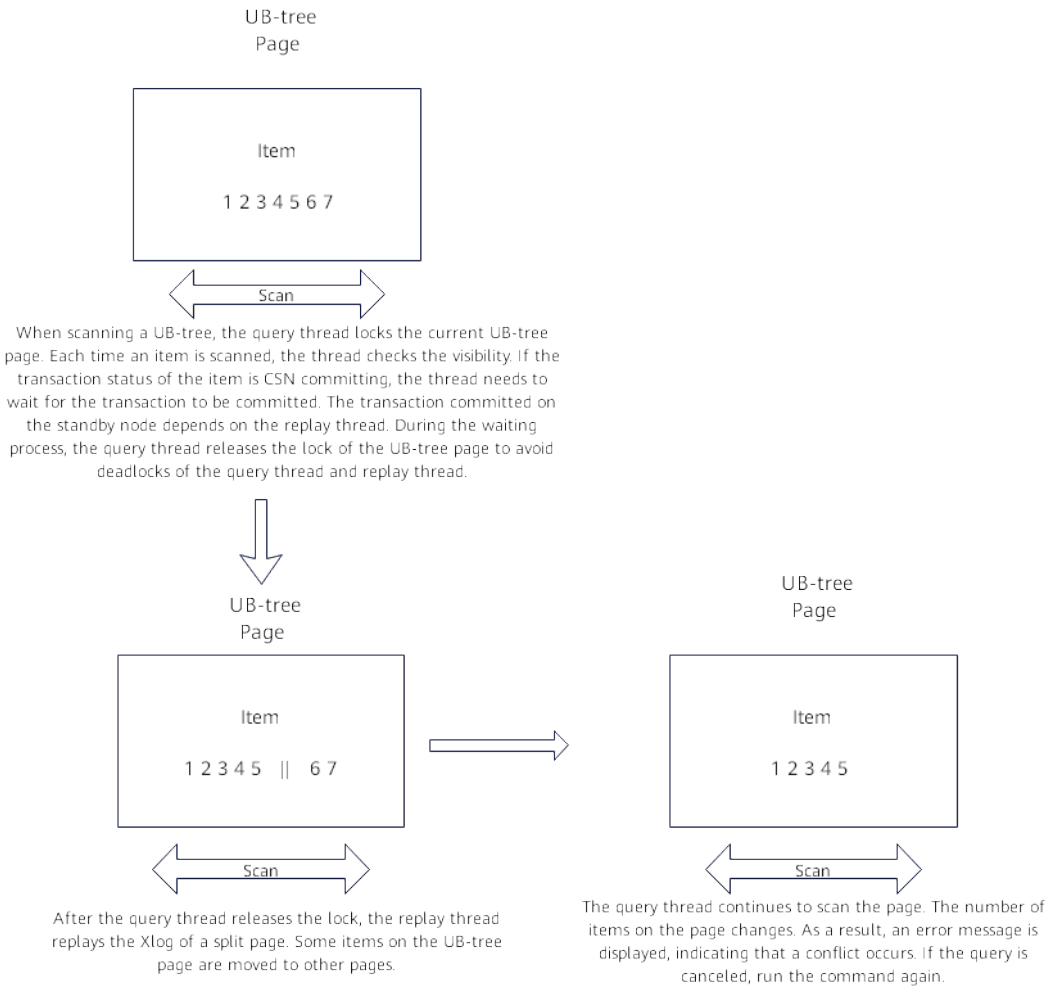
This error occurs only when the same index page needs to be accessed during query and replay. When the query thread releases the lock and waits for the transaction to end, the accessed page is modified. Figure 1 shows the process.

NOTE

- When scanning tuples in the committing state, the standby node needs to wait for transactions to be committed because the transaction committing sequence and log generation sequence may be out of order. For example, the transaction **tx_1** on the primary node is committed earlier than transaction **tx_2**, the commit log of **tx_1** on the standby node is replayed after the commit log of **tx_2**. According to the transaction committing sequence, **tx_1** should be visible to **tx_2**. Therefore, you need to wait for the transaction to be committed.
- When the standby node scans the index page, it is found that the number of tuples (including dead tuples) on the page changes and cannot be retried. This is because the scan may be forward or reverse scan. For example, after the page is split, some tuples are moved to the right page. In the case of reverse scan, even if the retry is performed, the tuples can only be read from the left, the correctness of the result cannot be ensured, and the split or insertion cannot be distinguished. Therefore, retry is not allowed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-18 Analysis



Solution

If an error is reported, you are advised to retry the query. In addition, you are advised to select index columns that are not frequently updated and use the soft deletion mode (physical deletion is performed during off-peak hours) to reduce the probability of this error.

7.9.3.6.4 Write Performance Deteriorates Occasionally When a Large Number of Concurrent Updates Are Performed During Long Query Execution

Symptom

When a table-level full scan long query is executed, a large number of concurrent page updates occur during the scan. As a result, the write performance of some DML statements deteriorates.

Analysis

For a long query (for example, more than two hours) in the full table scan scenario, before a page is scanned, a large number of concurrent updates (for

example, more than 100,000 updates) occur on the page. When the page is scanned later, a large number of historical versions need to be accessed to obtain visible tuples (based on the MVCC mechanism). Because a page read lock is held during single-page scan, if the page needs to be written at this time, the write is blocked until read on the page tuple is complete.

Troubleshooting

1. Check whether long queries and DML statements canceled due to timeout exist based on slow SQL alarms and the statement_history view.
2. Check the details of the canceled DML statements queried from statement_history in **1**, use the statement_detail_decode system function to parse the **details** field, and obtain the wait events. If the top wait event is BufferContentLock, there is a high probability that this problem occurs.

Solution

Prevention: Do not perform full table scan long queries on a table involving high-concurrency operations. You are advised to perform long queries on the standby node.

Handling: Check whether this scenario is triggered based on slow SQL alarms. You can interrupt long queries to avoid continuous impact on services.

7.9.4 Data Lifecycle Management: OLTP Table Compression

7.9.4.1 Introduction

OLTP table compression is a feature of GaussDB advanced compression.

This feature incorporates innovative technologies, including new compression algorithms, fine-grained automatic cold and hot determination, and intra-block compression. It enables OLTP table compression with a suitable compression ratio, significantly reducing the impact on services, enhancing background scheduling, improving job execution status queries, and saving space. It plays an important role in capacity control, supporting critical online services. The current version supports heap table compression, cross-page transparent compression, index compression, and compression of data batch imported by COPY. Additionally, you can configure the compression level and database-level compression policies to meet specific compression requirements for various service scenarios.

The advanced compression (ADVANCED) function is not recommended. If you need to use this function, contact Huawei technical support.

7.9.4.2 Limitations

For advanced compression:

- System catalogs, MOTs, global temporary tables, local temporary tables, sequence tables, Ustore segment-page tables, unlogged tables, and compressed TOAST data are not supported.
- You can set ILM policies for ordinary tables, partitions, and subpartitions.
- Only TDE tables and Astore/Ustore user tables of an ordinary table/partition/subpartition are supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- This feature is valid only in ORA-compatible, PG-compatible, or MySQL-compatible mode.
- Ustore does not support encoding and decoding, thus offering a compression ratio lower than Astore.
- The partial decompression feature has been added to the current version. Consequently, certain small tables are not compressed because their compression gain does not meet the requirements. However, they can be compressed in earlier versions without the partial decompression feature. For example, a table containing only one column of the int1 type cannot be compressed since the current version.
- In ORA-compatible mode, for the datea type, when the **mapping_date_to_datea** parameter is enabled, the compression ratio is affected due to changes in the storage structure of the data type. Compared to the date type, the compression ratio of the datea type may be lower.
- The GET_COMPRESSION_RATIO API does not support row-level expressions.
- In-place update cannot be performed on the row ID of a compressed table.
- After a compressed user table is converted to a rowid table, the table data is automatically decompressed.
- For rowid tables, the **rowid**/**rowno** system column cannot be used as a row-level expression condition.

For advanced index compression:

- Only B-tree index compression is supported in scenarios such as compression and background (manual or scheduled) scheduling, which involve build or rebuild index operations.
- During index build, only full compression is supported. Background scheduling performs compression based on the last modified time (LMT).
- When a compressed index needs to be rebuilt, you can only fully compress it, cancel the compression, or perform the compression based on the page LMT after the rebuild is complete.
- The current version only supports Astore and does not support Ustore.
- Index compression depends on policy information on the table or partition where the index is located.
- Indexes, secondary (local) indexes, and global indexes on Astore ordinary tables, partitioned tables, and hash bucket tables can be compressed. GSI compression is not supported.
- This feature is valid only in ORA-compatible, PG-compatible, or MySQL-compatible mode.
- In certain scenarios, compression of indexes may lead to an increase in occupied space. For example, when creating a global index and compressing it for a partitioned table and updating the global index during partition merging, new keys are inserted into the global index due to partition changes, potentially causing the global index to split. Consequently, the global index occupies more space after partition merging.
- For online index compression, the released pages cannot be reused immediately; they are actually released during the next round of scheduled compression after meeting certain conditions. When an index key is randomly inserted, it may be placed in a compressed page, leading to page splitting—a phenomenon more likely to occur in compressed pages, which typically

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

contain more index tuples than their uncompressed counterparts. Consequently, additional space may be required, resulting in potentially greater space bloat for compressed indexes compared to uncompressed ones. Given the context, enabling online index compression is not recommended when index keys are randomly inserted or when it is challenging to differentiate between cold and hot indexes. If index compression has been enabled for the index, you can delete the table policy to disable index compression (heap table compression is also disabled). Alternatively, change the value of **ENABLE_INDEX_COMPRESS** in gs_ilm_param to **0** to disable the index compression function (other indexes also cannot be compressed).

- When a table has multiple indexes, index compression must be enabled for either all of them or none; index compression cannot be controlled per index.
- This feature cannot be used together with index skip scan. If the index skip scan is enabled, an error is reported when the system scans compressed index rows.
- After upgrading from a database version that does not support index compression, to use the index compression feature in the original table, set a table policy for index compression to take effect.
- If posting has been configured for index deduplication and compression, ILM index compression is not supported during index build.
- After the upgrade from a database version that does not support index compression, if you want to use the index compression feature in the original table, set a table-level policy for the index compression to take effect.

For advanced compression of data batch import using COPY:

- Compression is supported during batch import using COPY or \COPY. Compression is not supported in the INSERT INTO SELECT scenario. Compression is also not supported during single-row import, for example, when you import a partitioned table with no rollback in the event of a constraint violation or a table with triggers.
- When used in conjunction with the error tolerance mechanism (LOG ERRORS or LOG ERRORS DATA), compression may not offer benefits if there are frequent occurrences of error data.
- TOAST data cannot be compressed during import.
- Because the compression of indexes occurs during build/rebuild or background scheduling, indexes are not compressed during batch import.
- This feature cannot be used during the upgrade observation period.
- Only Astore tables are supported. Ustore tables are not supported.
- This feature is supported in ORA-compatible, PG-compatible, and MySQL-compatible modes. It is not supported in M-compatible mode.
- This feature cannot be used together with logical decoding. If logical decoding is enabled, an error will occur during the data compression process when you import data using COPY.

For advanced database-level policies:

- Only advanced database-level compression policies are supported.
- Database-level policies are not applicable during the upgrade observation period.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Database-level policies do not support expression functionality.
- When using gs_dump to export database-level compression policies, ensure that the entire database is exported and you have the system administrator permission. If the preceding conditions are not met, the related policies are exported as independent policies at the table, partition, or subpartition level instead of being processed as database-level policies.

For cross-page transparent compression (TURBO):

- Only Astore/Ustore common row-store tables created by users in page mode are supported. An error is reported if other types of tables are created.
- Cross-page transparent compression does not apply to TOAST tables, and there is no improvement in compressing these tables.
- This feature is valid only in ORA-compatible, PG-compatible, or MySQL-compatible mode.
- Cross-page transparent compression cannot be used with other table compression algorithms. Doing so will result in an error.
- This feature cannot be used together with TDE. Doing so will result in an error.
- Pattern encoding only supports A-heap and U-heap pages; other page types are useless in improving the compression ratio.
- The ILM compression policies ADD, DELETE, and ADD+DELETE are not supported. Using these policies will result in an error.
- ILM policy type changes are not supported during partition splitting or merging. Making such changes will result in an error.
- ILM compression policies cannot be specified during REINDEX. Doing so will result in an error.
- During partition exchange, the cross-page transparent compression configurations (compress unit, compressor, and encoder) of the tables and partitions must be consistent; otherwise, an error is reported.
- Online DDL operations are not supported. If such operations are performed, they will be automatically executed in offline mode.
- Cross-page transparent compression and decompression incur significant costs and are only suitable for cold data.
- The syntax and scheduling mode of cross-page transparent compression follow the existing ILM mechanism and do not support new compatibility features.

For index compression (TURBO):

- Only data files under B-tree or UB-tree indexes created by users in page mode are supported. An error is reported if other types of indexes are created.
- The compression type of the indexes must be the same as those of the parent table. Otherwise, an error is reported.
- Indexes do not support configuration of cross-page transparent compression policies specifically for partitions. Otherwise, an error is reported.
- If an index compression policy is explicitly specified, the specified policy is used (the compression level is MEDIUM by default if no compression level is specified). If no index compression policy is explicitly specified, the index compression status and compression level are the same as those of the table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For the database-level policy (TURBO):

- Creation of cross-page transparent compression tables is not allowed during the upgrade observation period. Otherwise, an error is reported.
- ADIO does not apply to cross-page transparent compression objects. When ADIO is enabled, buffered I/O will still be used for disk read and write operations during cross-page compression.
- The block buffer size specified during backup does not apply to MAINFORK of cross-page transparent compression objects.

For the compression level:

- Medium-/High-level advanced compression is supported. If the compression level is not specified, the medium level is used by default with the LZ4 compression algorithm; if the high level is specified, the zstd compression algorithm is used.
- Low-/Medium-/High-level cross-page transparent compression is supported. If no level is specified, the default is medium.
- The compression level cannot be specified during the upgrade observation period.

7.9.4.3 Specifications

For advanced compression:

- If the TPC-C policy is enabled and scheduling is disabled, existing services are not affected.
- If the TPC-C compression policy is disabled, existing services are not affected.
- If an ILM policy is set for TPC-C table **bmsql_order_line** (only orders that have been delivered are identified as cold rows) and scheduling is disabled, the tpmC deterioration will not exceed 2% (taking a 56-core CPU, 370 GB memory, 3 TB SSD, and 350 GB shared buffer as an example).
- If an ILM policy is set for TPC-C table **bmsql_order_line** (only orders that have been delivered are identified as cold rows) and parameter scheduling is enabled by default, the tpmC deterioration will not exceed 5% (taking a 56-core CPU, 370 GB memory, 3 TB SSD, and 350 GB shared buffer as an example).
- The processing rate of a single-thread ILM job will be about 100 MB/s (taking a 56-core CPU, 370 GB memory, 3 TB SSD, and 350 GB shared buffer as an example).

  The processing rate can be measured based on the start time and end time of compression and the number of compressed pages.
- When GET is used to query data, the performance of accessing compressed data deteriorates compared to that of accessing non-compressed data. The performance deterioration on the driver side will not exceed 10%, and that on the PL/SQL side will not exceed 15% (taking a 32 MB shared buffer and 60,000 data pages as an example).
- When multi-get is used to query data, the performance of accessing compressed data deteriorates compared to that of accessing non-compressed data. The performance deterioration on the driver side will not exceed 30%, and that on the PL/SQL side will not exceed 40% (taking a 32 MB shared buffer and 60,000 data pages as an example).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When table-scan is used to query data, the performance of accessing compressed data deteriorates compared to that of accessing non-compressed data. The performance deterioration on the driver side will not exceed 30%, and that on the PL/SQL side will not exceed 40% (taking a 32 MB shared buffer and 60,000 data pages as an example).
- The compression ratio of the TPC-H table **lineitem** (all cold rows) is greater than or equal to 2:1.
- Tests on the TPC-C table **Orderline** and the TPC-H tables **Lineitem**, **Orders**, **Customer**, and **Part tables** show that the compression ratio is higher than that of LZ4 and ZLIB when there are many numeric columns; when there are a large number of text columns, the compression ratio is between that of the compression algorithms of the LZ class and the LZ+Huffman class.
- If a high-level compression policy is specified:
  - Compression ratio: Take the TPC-H table **lineitem** as an example. When Astore or Ustore is used, the compression ratio of the high level will be 10% higher than that of the medium level.
  - Performance: The ZSTD algorithm is suitable for historical service scenarios that are insensitive to performance, and the performance of accessing compressed rows deteriorates significantly. Using the TPC-C scenario as an example, with default compression parameter settings (including disabling metadata compression), the four transactional tables **bmsql_history**, **bmsql_new_order**, **bmsql_oorder**, and **bmsql_order_line** are all compressed. Subsequently, the TPC-C service is executed. Under conditions of 1000 warehouses and 500 concurrent tasks, the Astore performance may deteriorate by over 30%, whereas the Ustore performance may deteriorate by over 40%.
- For TPC-H table **lineitem** (**scale** set to **1**) and the TPC-C tables **bmsql_order_line**, **bmsql_history**, **bmsql_oorder**, and **bmsql_customer** (32 warehouses), if the default parameters are used and the tables are not compressed (the sampling rate is 20% and the sampled data volume is greater than 12 MB), the average error of the compression ratio predicted by the GET_COMPRESSION_RATIO API (all data is compressed) does not exceed 10%.
- In the sysbench scenario, when the default compression parameters are used and metadata compression is enabled, the performance of performing a read-only query with a filter containing row IDs decreases by no more than 70% compared with that without compression.

For advanced index compression:

- When the high-level compression policy is used, the average index compression and build time of the four transactional tables **bmsql_history**, **bmsql_new_order**, **bmsql_oorder**, and **bmsql_order_line** of the TPC-C standard model will decrease by no more than 50% in a single thread.
- When the medium-level compression policy is used, the average index compression and build time of the four transactional tables **bmsql_history**, **bmsql_new_order**, **bmsql_oorder**, and **bmsql_order_line** of the TPC-C standard model will decrease by no more than 30% in a single thread.
- When the high-level compression policy is used, index build compression is applied to the four transactional tables **bmsql_history**, **bmsql_new_order**, **bmsql_oorder**, and **bmsql_order_line** of the TPC-C standard model with a build compression ratio of 5:1.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When the medium-level compression policy is used, index build compression is applied to the four transactional tables **bmsql_history**, **bmsql_new_order**, **bmsql_oorder** and **bmsql_order_line** of the TPC-C standard model with a build compression ratio of 2.5:1.
- When the medium-level compression policy is used, for a configuration of 88-core CPU, 384 GB memory, and 64 TB disk capacity in the case of 500 concurrent queries in 1000 warehouses, after the heap tables and indexes of **bmsql_history**, **bmsql_new_order**, **bmsql_oorder tables**, and **bmsql_order_line** are compressed (with metadata compression disabled), the TPC-C performance will decrease by no more than 10% when compared to the performance when only heap tables are compressed.
- When the medium-level compression policy is used, for a configuration of 88-core CPU, 384 GB memory, and 64 TB disk capacity, after indexes and heap tables are compressed (with metadata compression disabled), the read-only and insert performance of SysBench will decrease by no more than 10%.

For compression of data batch import using COPY:

- Typical compression ratio: The compression ratios are related to data sets and data distribution. For instance, for the TPC-H table **lineitem**, the compression ratio for batch import via COPY and scheduling remains consistent at 2:1.
- Performance specifications (16-core CPU, 64 GB memory, with default compression settings, which disables metadata and index compression):
  a. Performance during import and compression:
     - Take the TPC-C and TPC-H scenarios as an example. When the default compression parameters are used (metadata compression is disabled), the compression performance (in terms of time consumption) will deteriorate by less than 20% during single-thread data import using COPY, compared to data import using COPY. (After the import, the index, TPC-H table **lineitem**, and the TPC-C tables **bmsql_history**, **bmsql_oorder**, and **bmsql_order_line** are created.)
     - For tables with a low compression ratio, the performance of importing such tables using COPY may deteriorate significantly due to the lack of I/O benefits from compression and increased CPU usage. For instance, the performance deterioration of the TPC-C table **bmsql_customer** could exceed 30%. Therefore, it is recommended that you do not perform COPY operations on such tables.

  b. Performance after import and compression: Consistent with scheduled compression. The four TPC-C transactional tables, **bmsql_history**, **bmsql_new_order**, **bmsql_oorder** and **bmsql_order_line**, are compressed during import and metadata and index compression are disabled. After full compression, TPC-C workloads are executed with a performance deterioration of no more than 10%, running 1000 warehouses with 500 concurrent tasks.
- The compression policy follows the specifications set by the COPY policy, superseding the table- or partition-level policy.
  – For a table, the COPY policy is applied, ignoring the ILM compression policy on the table. Compression and import are performed even if no ILM compression policy is applied to the table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - For a partition, the COPY policy is used as a guideline. Even if a partition does not have an ILM compression policy, data imported into that partition will still be compressed.
- The size after batch import and compression using COPY matches the filling factor of the page.
- During data import and compression using COPY, the encoding and compression algorithms are called, resulting in increased CPU computing power consumption. Additionally, since the data stream is divided into compressed and uncompressed streams, memory usage for each table or partition may increase by approximately 10 MB.

For cross-page transparent compression (TURBO):

- Function and resource specifications (medium-level compression where **compress unit** is set to **16** and **encoder** is set to **none** by default):
  - The compression ratio for typical financial data sets is approximately 4:1, whereas the equal-weighted compression ratio is around 5:1. The compression ratio for typical logistics data sets is roughly 6:1, whereas the equal-weighted compression ratio is about 7:1.
  - Memory resources are required for compression and decompression. When the thread pool feature is enabled, additional memory required is calculated as follows: Maximum number of concurrent sessions x 1 MB.
  - Managing compressed metadata requires additional memory resources, specifically 1/128 of the shared buffer size.
  - A doublewrite file will be created for each instance, with a maximum file size of 2 GB.
  - For the standby node, the JOB task limitations do not apply.
- The disk space usage of compressed tables may change in the following scenarios:
  - The settings of GUC parameter **turbo_compress_pattern_encoder** (different pattern encoders) are different on the primary and secondary nodes. Consequently, the size of the compressed tables on the primary and secondary nodes may vary.
  - The pattern encoder takes effect only in the ILM task compression scenario for performance reasons; it does not apply to other compression scenarios. Thus, after the pattern encoder is enabled, there may be a certain amount of storage space bloat when synchronous compression is triggered, which can be eliminated through the ILM compression task.
  - When logical scale-out is performed on the compressed table, because the pattern encoder is only applied in the ILM task compression scenario, the table data experiences a slight bloat after the logical scale-out (with the bloat rate not exceeding the compression ratio gain of the pattern encoder, typically around 30%). Currently, the scheduled ILM compression task can be used to eliminate bloat only after the scale-out is complete. Additionally, the value of **gsilmpolicy_seq changes** may change because the intermediate objects generated during the scale-out of the compressed table consume the sequence values.
- Cross-page compression performance estimation method:

  Typical large-capacity user scenario:
  1. CPU: 2P; memory: 1 TB; disk type: EVS (ultra-high I/O, 4 x striped disks) or SAS/SATA SSD (4 x RAID 0), with NVMe cache for acceleration

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. The read bandwidth of SATA SSDs is 500 MB/s, and that of SAS SSDs is 1 GB/s.
3. User data compression ratio: 5:1
4. Medium-level compression: **compress unit** is set to **16** and **encoder** is set to **none** by default

a. Cross-page compression/decompression increases I/O resource consumption and causes read/write amplification. The I/O resource consumption of a single read/write operation increases by three to four times.

b. Cross-page compression/decompression increases the CPU computing power consumption. For instance, according to the official benchmark specifications for zstd-level1, if the compression speed is 500 MB/s and the decompression speed is 1660 MB/s, in the default configuration of cross-page compression, the increase in the decompression read time of a single page can be calculated as follows:
Method 1: (Compression unit data size/Decompression speed) + (Compression unit compressed data size/Disk read rate)/(Single-page size/Disk read rate)

The following example is used for illustration purposes.

Example 1: (1) If the SAS SSD read bandwidth is 1 GB/s and the user data compression ratio is 5:1, the total time required for loading a single page increases by about 13 times. If the disk performance is good and the I/O time is close to the CPU time, the overall proportion of CPU decompression costs increases. In this case, the CPU computing power becomes a bottleneck. (2) If the SATA SSD read bandwidth is 500 MB/s and the user data compression ratio is 5:1, the time required for loading a single page increases by about eight times. In conclusion, the better the disk performance, the more obvious the deterioration of loading a single page.

Note: In real-world scenarios, service access is typically local, with extreme cold-loading scenarios being rare. Hence, the overall performance can be evaluated based on the actual test results of the TPC-C cold, warm, and hot models.

c. Cross-page compression causes performance deterioration during access to compressed MAINFORK files due to decompression and I/O amplification issues. The performance deterioration evaluation method is as follows:
Method 2: The impact of compression on performance can be evaluated as follows: Service volume of compressed pages (X%) x N (a multiplier for extra time required due to decompression and I/O amplification during the read of compressed pages)

d. Cross-page compression increases the time required to restore data from the doublewrite area during recovery. When backend dirty data flushing is enabled, the time increase is measured as follows:
Method 3: (Total I/O volume of data read from the doublewrite area + Total I/O volume of data written back to the disk)/I/O throughput. The total I/O volume is calculated as follows: 512 KB x Number of sessions written to the doublewrite area at the same time.

Note: The probability of concurrent sessions being simultaneously written to the doublewrite area in the cold, warm, and hot models is very low. The estimated impact on fault recovery time is at the millisecond level.

e. Cross-page compression leads to performance deterioration of both the primary and standby nodes during the log replay phase, negatively impacting the performance of read on standby, incremental build performance, as well as the incremental backup and restoration performance. For details about how to measure the time increased, refer to **c**.

7.9.4.4 Usage Guide

To use the advanced compression function, you must obtain a license. For more information, please contact Huawei technical support.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 1 Enable the compression function:

```
gaussdb=# ALTER DATABASE SET ilm = on;
```

> **NOTE**
>
> To run the **ALTER DATABASE SET ilm = on** command, you must have the PUBLIC permission.

Check whether **gsilmpolicy_seq** and **gsilmtask_seq** exist in the public schema of the current database.

```
gaussdb=# \d
                        List of relations
 Schema |      Name       |   Type   |   Owner    |                 Storage
--------+-----------------+----------+------------+---------------------------------------------------------
 public | gsilmpolicy_seq | sequence | omm |
 public | gsilmtask_seq   | sequence | omm |
```

or

```
gaussdb=# SELECT a.oid, a.relname FROM pg_class a inner join pg_namespace b on a.relnamespace = b.oid
   WHERE (a.relname = 'gsilmpolicy_seq' OR a.relname = 'gsilmtask_seq') AND b.nspname = 'public';

 oid  |     relname
-------+-----------------
 17002 | gsilmpolicy_seq
 17004 | gsilmtask_seq
(2 rows)
```

If an exception occurs, the system reports a warning.

```
WARNING:  ILM sequences are already existed while initializing
```

Step 2 Add compression policies for a table.

- Create a table with a policy.

  ```
  gaussdb=# CREATE TABLE ilm_table_1 (col1 int, col2 text)
     ilm add policy row store compress advanced row
     after 3 days of no modification on (col1 < 1000);
  ```

- Add a policy for an existing table.

  ```
  gaussdb=# CREATE TABLE ilm_table_2 (col1 int, col2 text);
  gaussdb=# ALTER TABLE ilm_table_2 ilm add policy row store
     compress advanced row after 3 days of no modification;
  ```

- Check whether data is added to the policy view.

  ```
  gaussdb=# SELECT * FROM gs_my_ilmpolicies;

   policy_name |  policy_type  | tablespace | enabled | deleted
  -------------+---------------+------------+---------+---------
   p1          | DATA MOVEMENT |            | YES     | NO
   p2          | DATA MOVEMENT |            | YES     | NO
  (2 rows)
  ```

- Check whether the policy that meets the settings is added to the policy details view.

  ```
  gaussdb=# SELECT * FROM gs_my_ilmdatamovementpolicies;

   policy_name | action_type | scope | compression_level | tier_tablespace | tier_status |
  condition_type     | condition_days | custom_function | policy_subtype | action_clause | tier_to
  -------------+-------------+-------+-------------------+-----------------+-------------+-------------------------
  +----------------+-----------------+----------------+---------------+---------
   p1          | COMPRESSION | ROW   | ADVANCED          |                 |             | LAST MODIFICATION
  TIME |              3 |                |                |             |
   p2          | COMPRESSION | ROW   | ADVANCED          |                 |             | LAST MODIFICATION
  TIME |              3 |                |                |             |
  (2 rows)
  ```

- Check whether the policy corresponds to the target table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# SELECT * FROM gs_my_ilmobjects;
 policy_name | object_owner | object_name | subobject_name | object_type |    inherited_from    |
tbs_inherited_from | enabled | deleted
-------------+--------------+-------------+----------------+-------------+----------------------
+--------------------+---------+---------
 p1          | public       | ilm_table_1 |                | TABLE       | POLICY NOT INHERITED |                    |
YES     | NO
 p2          | public       | ilm_table_2 |                | TABLE       | POLICY NOT INHERITED |                    |
YES     | NO
(2 rows)
```

- Create a table with a high-level compression policy.

```
gaussdb=# CREATE TABLE ilm_table_h1 (col1 int, col2 text)
    ilm add policy row store compress advanced high row
    after 3 days of no modification on (col1 < 1000);
```

- Add a high-level compression policy for the inventory table.

```
gaussdb=# CREATE TABLE ilm_table_h2 (col1 int, col2 text);
gaussdb=# ALTER TABLE ilm_table_h2 ilm add policy row store
    compress advanced high row after 3 days of no modification;
```

Step 3 Perform compression evaluation.

- Manually perform compression evaluation.

  You can change the value of **POLICY_TIME** to determine whether the evaluation time is in days or seconds. Only the system administrators can change the value. For details, see DBE_ILM_ADMIN.CUSTOMIZE_ILM in "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_ILM_ADMIN" in *Reference*.

  Insert random data for testing.

```
gaussdb=# INSERT INTO ilm_table_1 select *, 'test_data' FROM generate_series(1, 10000);

gaussdb=# DECLARE
    v_taskid number;
gaussdb=# BEGIN
    DBE_ILM.EXECUTE_ILM(OWNER           => 'public',
                        OBJECT_NAME     => 'ilm_table_1',
                        TASK_ID         => v_taskid,
                        SUBOBJECT_NAME  => NULL,
                        POLICY_NAME     => 'ALL POLICIES',
                        EXECUTION_MODE  => 2);
    RAISE INFO 'Task ID is:%', v_taskid;
gaussdb=# END;
/
```

  If the input parameter is incorrect, the system reports an error. If no error occurs, the system does not display any information. (The RAISE INFO statement is added to the preceding code segment to print the ID of the current task.)

```
INFO:  Task ID is:1
```

  Check the task information.

```
gaussdb=# SELECT * FROM gs_my_ilmtasks;

task_id | task_owner |   state   |         creation_time         |          start_time           |
completion_time
---------+------------+-----------+-------------------------------+-------------------------------
+-------------------------------
       1 | omm| COMPLETED | 2023-08-29 17:36:38.779555+08 | 2023-08-29 17:36:38.779555+08 |
2023-08-29 17:36:38.879485+08
(1 row)
```

  Check the evaluation result.

```
gaussdb=# SELECT * FROM gs_my_ilmevaluationdetails;

 task_id | policy_name | object_owner | object_name | subobject_name | object_type |
selected_for_execution |      job_name      | comments
---------+-------------+--------------+-------------+----------------+-------------+------------------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
+-------------------+----------
      1 | p1          | public       | ilm_table_1 |               | TABLE       | SELECTED FOR EXECUTION | ilmjob
$_postgres1 |
(1 row)
```

Check the compression job information.

```
gaussdb=# SELECT * FROM gs_my_ilmresults;

 task_id |     job_name      |       job_state        |          start_time           |        completion_time        |
comments |               statistics
---------+-------------------+------------------------+-------------------------------
+-------------------------------+----------+---------------------------------------
      1 | ilmjob$_postgres1 | COMPLETED SUCCESSFULLY | 2023-08-29 17:36:38.779555+08 |
2023-08-29 17:36:38.879485+08 |          | SpaceSaving=0,BoundTime=0,LastBlkNum=0
(1 row)
```

- Trigger automatic scheduling evaluation in the background.

  Log in to the **template1** database as the initial user and create a maintenance window.

```
gaussdb=# DECLARE
   V_HOUR INT := 22;
   V_MINUTE INT := 0;
   V_SECOND INT := 0;
   C_ADO_WINDOW_SCHEDULE_NAME TEXT := 'ado_window_schedule';
   C_ADO_WINDOW_PROGRAM_NAME TEXT := 'ado_window_program';
   C_MAINTENANCE_WINDOW_JOB_NAME TEXT := 'maintenance_window_job';
   V_MAINTENCE_WINDOW_REPEAT TEXT;
   V_MAINTENCE_WINDOW_START TIMESTAMPTZ;
   V_BE_SCHEDULE_ENABLE BOOL;
   V_MAINTENANCE_WINDOW_EXIST INT;
BEGIN
   SELECT COUNT(*) INTO V_MAINTENANCE_WINDOW_EXIST FROM PG_CATALOG.PG_JOB WHERE
JOB_NAME = 'maintenance_window_job' AND DBNAME = 'template1';
   IF CURRENT_DATABASE() != 'template1' THEN
      RAISE EXCEPTION 'Create maintenance_window FAILED, current database is not template1';
   END IF;
   IF V_MAINTENANCE_WINDOW_EXIST = 0 AND CURRENT_DATABASE() = 'template1' THEN
      SELECT
         CASE
            WHEN NOW() < CURRENT_DATE + INTERVAL '22 HOUR' THEN CURRENT_DATE +
INTERVAL '22 HOUR'
            ELSE CURRENT_DATE + INTERVAL '1 DAY 22 HOUR'
         END INTO V_MAINTENCE_WINDOW_START;
      --1. prepare for maintenance window schedule
      SELECT 'freq=daily;interval=1;byhour='||V_HOUR||';byminute='||V_MINUTE||';bysecond='||
V_SECOND INTO V_MAINTENCE_WINDOW_REPEAT;
      BEGIN
         SELECT
            CASE
               WHEN VALUE = 1 THEN TRUE -- DBE_ILM_ADMIN.ILM_ENABLED
               ELSE FALSE
            END INTO V_BE_SCHEDULE_ENABLE
            FROM PG_CATALOG.GS_ILM_PARAM WHERE IDX = 7; -- DBE_ILM_ADMIN.ENABLED
      EXCEPTION
         WHEN OTHERS THEN
            V_BE_SCHEDULE_ENABLE := FALSE;
      END;
      --2. Create ado window schedule
      DBE_SCHEDULER.CREATE_SCHEDULE(
         SCHEDULE_NAME => C_ADO_WINDOW_SCHEDULE_NAME,
         START_DATE => '9999-01-01 00:00:01',
         REPEAT_INTERVAL => NULL,
         END_DATE => NULL,
         COMMENTS => 'ado window schedule');
      --3. Create ado window program
      DBE_SCHEDULER.CREATE_PROGRAM(
         PROGRAM_NAME => C_ADO_WINDOW_PROGRAM_NAME,
         PROGRAM_TYPE => 'plsql_block',
         PROGRAM_ACTION => 'call prvt_ilm.be_execute_ilm(0);',
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        NUMBER_OF_ARGUMENTS => 0,
        ENABLED => TRUE,
        COMMENTS => NULL);
    --4. Create maintenance window master job
    DBE_SCHEDULER.CREATE_JOB(
        JOB_NAME => C_MAINTENANCE_WINDOW_JOB_NAME,
        START_DATE => V_MAINTENCE_WINDOW_START,
        REPEAT_INTERVAL => V_MAINTENCE_WINDOW_REPEAT,
        END_DATE => NULL,
        JOB_TYPE => 'STORED_PROCEDURE'::TEXT,
        JOB_ACTION => 'prvt_ilm.be_active_ado_window'::TEXT,
        NUMBER_OF_ARGUMENTS => 0,
        ENABLED => V_BE_SCHEDULE_ENABLE,
        AUTO_DROP => FALSE,
        COMMENTS => 'maintenance window job',
        destination_name=> 'CCN');
  ELSE
    RAISE EXCEPTION 'CREATE ILM MAINTENANCE WINDOW FAILED';
  END IF;
END;
```

Automatic scheduling provides the following parameters for adjustment:

```
gaussdb=# SELECT * FROM gs_adm_ilmparameters;
             name              | value
-----------------------------------+-------
 EXECUTION_INTERVAL              |    15
 RETENTION_TIME                  |    30
 ENABLED                         |     1
 POLICY_TIME                     |     0
 ABS_JOBLIMIT                    |    10
 JOB_SIZELIMIT                   |  1024
 WIND_DURATION                   |   240
 BLOCK_LIMITS                    |    40
 ENABLE_META_COMPRESSION         |     0
 SAMPLE_MIN                      |    10
 SAMPLE_MAX                      |    10
 CONST_PRIO                      |    40
 CONST_THRESHOLD                 |    90
 EQVALUE_PRIO                    |    60
 EQVALUE_THRESHOLD               |    80
 ENABLE_DELTA_ENCODE_SWITCH      |     1
 LZ4_COMPRESSION_LEVEL           |     0
 ENABLE_LZ4_PARTIAL_DECOMPRESSION |     1
 INDEXJOB_SIZELIMIT              |  1024
 ENABLE_INDEX_COMPRESS           |     0
(20 rows)
```

- **EXECUTION_INTERVAL**: interval for executing the automatic scheduling task. By default, the task is executed every 15 minutes.
- **RETENTION_TIME**: interval for deleting historical compression task records. By default, historical compression task records are deleted every 30 days.
- **ENABLED**: specifies whether automatic scheduling is enabled. The default value is **ENABLED**.
- **POLICY_TIME**: time unit for policy evaluation, which is used for tests. The default unit is days.
- **ABS_JOBLIMIT**: maximum number of compression tasks generated in a single evaluation. The default value is **10**.
- **JOB_SIZELIMIT**: I/O upper limit of a single compression task. The default value is 1 GB.
- **WIND_DURATION**: duration of a single maintenance window.
- **BLOCK_LIMITS**: upper limit of the instance-level row-store compression speed. The default value is **40**. The value ranges from 0 to 10000, in

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

blocks/ms, indicating the maximum number of blocks that can be compressed per millisecond. The value **0** indicates that there is no speed limit. The maximum speed is calculated as follows: **BLOCK_LIMITS** x 1000 x **BLOCKSIZE**. For example, if the default value is **40**, the maximum speed is 320,000 KB/s (40 x 1000 x 8 KB).

- **ENABLE_META_COMPRESSION**: specifies whether header compression is enabled. The default value is **0**. The value can be **0** (disabled) or **1** (enabled).

  NOTE

  If this parameter is set to **1**, the compression ratio is improved for tables with short rows (for example, less data in a single row), but the performance of accessing compressed rows significantly deteriorates. If most tables in the database have long rows, it is recommended that you disable this parameter.

- **SAMPLE_MIN**: minimum sampling step for constant encoding and equivalent encoding. The default value is **10**. The value range is [1,100]. Decimals will be automatically rounded down to the nearest lower value.
- **SAMPLE_MAX**: maximum sampling step for constant encoding and equivalent encoding. The default value is **10**. The value range is [1,100]. Decimals will be automatically rounded down to the nearest lower value.
- **CONST_PRIO**: constant encoding priority. The default value is **40**. The value range is [0,100]. The value **100** indicates that constant encoding is disabled. Decimals will be automatically rounded down to the nearest lower value.
- **CONST_THRESHOLD**: constant encoding threshold. The default value is **90**. The value range is [1,100], indicating that constant encoding is performed when the proportion of constant values in a column exceeds the threshold. Decimals will be automatically rounded down to the nearest lower value.
- **EQVALUE_PRIO**: equivalent encoding priority. The default value is **60**. The value range is [0,100]. The value **100** indicates that equivalent encoding is disabled. Decimals will be automatically rounded down to the nearest lower value.
- **EQVALUE_THRESHOLD**: equivalent encoding threshold. The default value is **80**. The value range is [1,100], indicating that equivalent encoding is performed when the ratio of equal values in two columns exceeds the threshold. Decimals will be automatically rounded down to the nearest lower value.
- **ENABLE_DELTA_ENCODE_SWITCH**: specifies whether difference encoding is enabled. The default value is **1**. The value **0** indicates that difference encoding is disabled, and the value **1** indicates that difference encoding is enabled. Decimals will be automatically rounded down to the nearest lower value.
- **LZ4_COMPRESSION_LEVEL**: LZ4 compression level. The default value is **0**. The value range is [0,16]. Decimals will be automatically rounded down to the nearest lower value.
- **ENABLE_LZ4_PARTIAL_DECOMPRESSION**: specifies whether partial decompression is enabled. The default value is **1**. The value **0** indicates that partial decompression is disabled, and the value **1** indicates that partial decompression is enabled. Decimals will be automatically rounded down to the nearest lower value.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **INDEXJOB_SIZELIMIT**: maximum number of bytes that can be processed by a single ADO job during index compression. The unit is MB. The default value is **1024**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
- **ENABLE_INDEX_COMPRESS**: specifies whether to enable index compression. The value **0** (default value) indicates that index compression is disabled, and the value **1** indicates that index compression is enabled.

The preceding parameters can be adjusted through the DBE_ILM_ADMIN.CUSTOMIZE_ILM() API.

By default, the maintenance window opens at 22:00 Beijing time every day. You can use the SET_ATTRIBUTE API provided by the DBE_SCHEDULER to set the maintenance window.

```
\c template1
 CALL DBE_ILM_ADMIN.DISABLE_ILM();
 CALL DBE_ILM_ADMIN.ENABLE_ILM();
DECLARE
    newtime timestamptz := CLOCK_TIMESTAMP() + to_interval('2 seconds');
BEGIN
    DBE_SCHEDULER.set_attribute(
        name          =>          'maintenance_window_job',
        attribute     =>          'start_date',
        value         =>          TO_CHAR(newtime, 'YYYY-MM-DD HH24:MI:SS')
    );
END;
/
```

----End

7.9.4.5 Setting the Maintenance Window Parameters

- **RETENTION_TIME**: duration for storing evaluation and compression records, in days. The default value is **30**. You can adjust the value based on your storage capacity.
- **EXECUTION_INTERVAL**: execution frequency of an evaluation task, in minutes. The default value is **15**. You can adjust the value based on the service and resource usage during the maintenance window. This parameter and **ABS_JOBLIMIT** affect each other. A single thread can generate a maximum of **WIND_DURATION/EXECUTION_INTERVAL** x **JOB_SIZELIMIT** I/Os per day.
- **JOB_SIZELIMIT**: maximum number of bytes that can be processed by a single compression job. The unit is MB. The default value is **1024**. The compression speed is about 100 MB/s. When the I/O limit of each compression job is 1 GB, the compression job can be completed within 10 seconds. You can adjust the value based on the service off-peak hours and the amount of data to be compressed.
- **ABS_JOBLIMIT**: maximum number of compression jobs that can be generated in an evaluation. You can adjust the value based on the number of partitions and tables in the configured policy. It is recommended that the value be less than or equal to 10. You can run the **select count(*) from gs_adm_ilmobjects where enabled = true** command to query the value.
- **POLICY_TIME**: specifies whether the time unit for determining cold rows is day or second. The time unit second is used only for testing. The value can be **ILM_POLICY_IN_SECONDS** or **ILM_POLICY_IN_DAYS** (default value).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **WIND_DURATION**: maintenance window duration, in minutes. The default value is 240 minutes (4 hours). By default, the maintenance window lasts 240 minutes from 22:00 (Beijing time). You can adjust the maintenance window based on service off-peak hours.

  The following parameters support only advanced compression (ADVANCED) and do not support cross-page transparent compression (TURBO).

- **BLOCK_LIMITS**: upper limit of the instance-level row-store compression speed. The default value is **40**. The value ranges from 0 to 10000, in block/ms, indicating the maximum number of blocks that can be compressed per millisecond. **0** indicates that there is no speed limit. The maximum speed is calculated as follows: **BLOCK_LIMITS** x 1000 x **BLOCKSIZE**. For example, if the default value is **40**, the maximum speed is 320,000 KB/s (40 x 1000 x 8 KB).
- **ENABLE_META_COMPRESSION**: specifies whether to enable header compression. The default value is **0**. The value can be **0** (disabled) or **1** (enabled). You can enable or disable this function as required.
- **SAMPLE_MIN**: minimum sampling step for constant encoding and equivalent encoding. The default value is **10**. The value range is [1,100]. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **SAMPLE_MAX**: maximum sampling step for constant encoding and equivalent encoding. The default value is **10**. The value range is [1,100]. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **CONST_PRIO**: constant encoding priority. The default value is **40**. The value range is [0,100]. The value **100** indicates that constant encoding is disabled. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **CONST_THRESHOLD**: constant encoding threshold. The default value is **90**. The value range is [1,100], indicating that constant encoding is performed when the proportion of constant values in a column exceeds the threshold. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **EQVALUE_PRIO**: equivalent encoding priority. The default value is **60**. The value range is [0,100]. The value **100** indicates that equivalent encoding is disabled. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **EQVALUE_THRESHOLD**: equivalent encoding threshold. The default value is **80**. The value range is [1,100], indicating that equivalent encoding is performed when the ratio of equal values in two columns exceeds the threshold. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **ENABLE_DELTA_ENCODE_SWITCH**: specifies whether to enable difference encoding. The default value is **1**. The value **0** indicates that difference encoding is disabled, and the value **1** indicates that difference encoding is enabled. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.
- **LZ4_COMPRESSION_LEVEL**: LZ4 compression level. The default value is **0**. The value range is [0,16]. Decimals can be entered and will be automatically rounded down. You can set this parameter as required.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **ENABLE_LZ4_PARTIAL_DECOMPRESSION**: specifies whether to enable partial decompression. The default value is **1**. The value **0** indicates that partial decompression is disabled, and the value **1** indicates that partial decompression is enabled. Decimals can be entered and will be automatically rounded down. You can enable or disable this function as required.
- **INDEXJOB_SIZELIMIT**: maximum number of bytes that can be processed by a single ADO job during index compression. The unit is MB. The default value is **1024**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
- **ENABLE_INDEX_COMPRESS**: specifies whether to enable index compression. The value **0** (default value) indicates that index compression is disabled, and the value **1** indicates that index compression is enabled.

Example:

```
EXECUTION_INTERVAL: 15
JOB_SIZELIMIT: 10240
WIND_DURATION: 240
BLOCK_LIMITS: 0
```

In this configuration, for a single table partition, a total of 160 GB (240/15 x 10240 MB) data can be evaluated and compressed during a maintenance window. The compression speed is 100 MB/s, but the actual compression takes only 27 minutes (160 GB/(100 MB/s) = 27 minutes). Services are not affected in other periods. You can adjust the parameters based on the available compression period during off-peak hours.

7.9.4.6 O&M Tips

NOTICE

Before using the advanced compression feature, contact Huawei technical support to request a license. Otherwise, an error will be reported when you run related commands.

1. Manually trigger compression once. (In the example, 102400 MB data is compressed once.)
   a. Add the cold and hot separation policy for a table.

```
gaussdb=#  DROP TABLE IF EXISTS ILM_TABLE;
gaussdb=#  CREATE TABLE ILM_TABLE(a int);
gaussdb=# ALTER TABLE ILM_TABLE ILM ADD POLICY ROW STORE COMPRESS ADVANCED
    ROW AFTER 3 MONTHS OF NO MODIFICATION;
```

   b. Manually trigger compression.

```
DECLARE
  v_taskid number;
BEGIN
  DBE_ILM_ADMIN.CUSTOMIZE_ILM(11, 1);
DBE_ILM_ADMIN.CUSTOMIZE_ILM(13, 102400);
  DBE_ILM.EXECUTE_ILM(OWNER        => '$schema_name',
                      OBJECT_NAME    => 'ilm_table',
                      TASK_ID        => v_taskid,
                      SUBOBJECT_NAME => NULL,
                      POLICY_NAME    => 'ALL POLICIES',
                      EXECUTION_MODE => 2);
  RAISE INFO 'Task ID is:%', v_taskid;
END;
/
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

c. Check whether the compression job is complete. You can view the detailed execution information.

```
gaussdb=# SELECT * FROM gs_adm_ilmresults order by task_id desc limit 1;
 task_id |                    job_name                    |       job_state        |          start_time
|        completion_time        | comments |                              statistics                              |
node_name

---------+------------------------------------------------+------------------------
+-------------------------------+--------------------------------+----------
+-----------------------------------------------------------------------------+-------------
N/A
   20004 | ilmjob$_postgres_test_20250302062047417126137 | COMPLETED SUCCESSFULLY |
2025-03-06 20:47:41.980633+08 | 2025-03-06 20:47:59.136747+08 |          |
SpaceSaving=98503180,BoundTime=1741265260,StartBlkNum=0,LastBlkNum=40709 |
gs_datanode3
(1 row)
```

d. Manually trigger index compression.

```
gaussdb=# CREATE INDEX ILM_INDEX ON ILM_TABLE(a);
gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(27, 1);

-- insert ticker function
CREATE OR REPLACE FUNCTION hex_to_int(hexval varchar) RETURNS bigint AS $$
    DECLARE
        result bigint;
    BEGIN
        EXECUTE 'SELECT x' || quote_literal(hexval) || '::bigint' INTO result;
        RETURN result;
    END;
$$ LANGUAGE plpgsql;
create or replace procedure insert_ticker()
as
    declare
        current_lsn varchar;
        temp_res text[];
        part_1 bigint;
        part_2 bigint;
        transformed_lsn bigint;
    begin
        select pg_current_xlog_location() into current_lsn;
        select string_to_array(current_lsn, '/') INTO temp_res;
        part_1 = hex_to_int(temp_res[1]);
        part_2 = hex_to_int(temp_res[2]);
        transformed_lsn = part_1 * 2 ^ 32 + part_2;
        insert into gs_ilm_ticker VALUES(transformed_lsn, CURRENT_TIMESTAMP - interval '30
seconds');
    end;
/

gaussdb=# CALL insert_ticker();
 insert_ticker
---------------

(1 row)

DECLARE
  v_taskid number;
BEGIN
  DBE_ILM_ADMIN.CUSTOMIZE_ILM(11, 1);
  DBE_ILM_ADMIN.CUSTOMIZE_ILM(26, 102400);
  DBE_ILM.EXECUTE_ILM(OWNER        => '$schema_name',
                      OBJECT_NAME    => 'ilm_index',
                      TASK_ID        => v_taskid,
                      SUBOBJECT_NAME => NULL,
                      POLICY_NAME    => 'ALL POLICIES',
                      EXECUTION_MODE => 2);
  RAISE INFO 'Task ID is:%', v_taskid;
END;
/
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=#  SELECT * FROM gs_adm_ilmresults order by task_id desc;
 task_id |                                job_name                                |       job_state        |
start_time           |        completion_time        | comments |
statistics
                                     |  node_name
---------+-----------------------------------------------------------------------------
+------------------------+-------------------------------+-------------------------------+----------
+-----------------------------------------------------------------
-------------------------------------------------+--------------
      37 | ilmindexjob$_postgres_test_idx3_202503041713161816797               | COMPLETED
SUCCESSFULLY | 2025-03-04 17:13:16.412746+08 | 2025-03-04 17:13:16.423239+08 |          |
ScanBlocks=32,CompressBlocks=0,MergeBlocks=0,RecycleBlocks=97,N
extScanBeginBlkNum=0                          | gs_datanode1
```

e. If **ENABLE_INDEX_COMPRESS** is set to **1** and an ILM policy has been added to the table, index build compression is triggered when **CREATE INDEX** is executed on the table to reduce the number of pages generated during index build.

2. Manually stop compression.

```
gaussdb=# CALL DBE_ILM.STOP_ILM (1,  FALSE, 'test_job'); -- task_id is 1, p_drop_running_Jobs is
FALSE, and p_Jobname is 'test_job'.
```

Table 7-12 DBE_ILM.STOP_ILM input parameters

Name	Description
TASK_ID	Specifies the descriptor ID of the ADO task to be stopped.
P_DROP_RUNNING_JOBS	Determines whether to stop a task that is being executed. The value **true** indicates that the task is forcibly stopped, and the value **false** indicates that the task is not stopped.
P_JOBNAME	Specifies the name of the job to be stopped, which can be queried in the GS_MY_ILMEVALUATIONDETAILS view.

3. Generate a policy for a table and schedule compression tasks in the background.

   a. Add the cold and hot separation policy for a table.

   ```
   gaussdb=# DROP TABLE IF EXISTS ILM_TABLE;
   gaussdb=# CREATE TABLE ILM_TABLE(a int);
   gaussdb=# ALTER TABLE ILM_TABLE ILM ADD POLICY ROW STORE COMPRESS ADVANCED
      ROW AFTER 3 MONTHS OF NO MODIFICATION;
   ```

   b. Set the parameters related to ILM execution.

   ```
   BEGIN
     DBE_ILM_ADMIN.CUSTOMIZE_ILM(11, 1);
     DBE_ILM_ADMIN.CUSTOMIZE_ILM(12, 10);
     DBE_ILM_ADMIN.CUSTOMIZE_ILM(1, 1);
     DBE_ILM_ADMIN.CUSTOMIZE_ILM(13, 512);
   END;
   /
   ```

   c. Enable scheduled scheduling in the background.

   ```
   gaussdb=# CALL DBE_ILM_ADMIN.DISABLE_ILM();
   gaussdb=# CALL DBE_ILM_ADMIN.ENABLE_ILM();
   ```

   d. You can call DBE_SCHEDULER.set_attribute to set the opening time of the maintenance window as required. By default, this function is enabled at 22:00.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

e. To enable the automatic scheduling task for index compression, set **ENABLE_INDEX_COMPRESS** to **1** by executing **DBE_ILM_ADMIN.CUSTOMIZE_ILM(27, 1)**.

4. Set the parameters related to ILM execution.

   Specifies whether the time unit of ADO is day or second. The time unit second is used only for testing. The setting can be **ILM_POLICY_IN_SECONDS = 1** or **ILM_POLICY_IN_DAYS = 0** (default value).

   ```
   gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(11, 1);
   ```

   Specifies the maximum number of ADO jobs generated by an ADO task. The value is an integer or floating-point number greater than or equal to 0 and less than or equal to 2147483647. The value is rounded down.

   ```
   gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(12, 10);
   ```

   Specifies the frequency of executing an ADO task, in minutes. The default value is **15**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.

   ```
   gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(1, 1);
   ```

   Specifies the maximum number of bytes that can be processed by a single ADO job. The unit is MB. The value is an integer or floating-point number greater than or equal to 0 and less than or equal to 2147483647. The value is rounded down.

   ```
   gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(13, 512);
   ```

5. Check whether a table is suitable for compression and evaluate the compression benefits. (Index compression is not supported.)

   ```
   DBE_COMPRESSION.GET_COMPRESSION_RATIO(
   SCRATCHTBSNAME IN VARCHAR2,
   OWNNAME        IN VARCHAR2,
   OBJNAME        IN VARCHAR2,
   SUBOBJNAME     IN VARCHAR2,
   COMPTYPE       IN NUMBER,
   BLKCNT_CMP     OUT INTEGER,
   BLKCNT_UNCMP   OUT INTEGER,
   ROW_CMP        OUT INTEGER,
   ROW_UNCMP      OUT INTEGER,
   CMP_RATIO      OUT NUMBER,
   COMPTYPE_STR   OUT VARCHAR2,
   SAMPLE_RATIO   IN NUMBER DEFAULT 20,
   OBJTYPE        IN INTEGER DEFAULT 1)
   ```

Table 7-13 Input parameters of DBE_COMPRESSION.GET_COMPRESSION_RATIO

Name	Description
SCRATCHTBSNAME	Tablespace to which a data object belongs.
OWNNAME	Data object owner (schema to which the data object belongs).
OBJNAME	Data object name.
SUBOBJNAME	Name of a data subobject.
COMPTYPE	• **1**: uncompressed. • **2**: advanced compression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description
SAMPLE_RATIO	Sampling ratio. The value is an integer or floating-point number ranging from 0 to 100, corresponding to the sampling ratio of *N* percent. The default value is **20**, indicating that 20% of the rows are sampled.
OBJTYPE	Object type. The options are as follows: **1**: table object.

Table 7-14 Output parameters of DBE_COMPRESSION.GET_COMPRESSION_RATIO

Name	Description
BLKCNT_CMP	Number of blocks occupied by compressed samples.
BLKCNT_UNCMP	Number of blocks occupied by uncompressed samples.
ROW_CMP	Number of rows that can be contained in a single block after samples are compressed.
ROW_UNCMP	Number of rows that can be contained in a single block when samples are not compressed.
CMP_RATIO	Compression ratio, that is, **blkcnt_uncmp** divided by **blkcnt_cmp**.
COMPTYPE_STR	Character string that describes the compression type.

Example:

```
gaussdb=# ALTER DATABASE set ilm = on;
gaussdb=# CREATE user user1 IDENTIFIED BY '********';
gaussdb=# CREATE user user2 IDENTIFIED BY '********';
gaussdb=# SET ROLE user1 PASSWORD '********';
gaussdb=# CREATE TABLE TEST_DATA (ORDER_ID INT, GOODS_NAME TEXT, CREATE_TIME
TIMESTAMP)
  ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW AFTER 1 DAYS OF NO MODIFICATION;
INSERT INTO TEST_DATA VALUES (1,'Snack package A', NOW());

DECLARE
o_blkcnt_cmp       integer;
o_blkcnt_uncmp     integer;
o_row_cmp          integer;
o_row_uncmp        integer;
o_cmp_ratio        number;
o_comptype_str     varchar2;
begin
dbe_compression.get_compression_ratio(
    SCRATCHTBSNAME  =>  NULL,
    OWNNAME         =>  'user1',
    OBJNAME         =>  'test_data',
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  SUBOBJNAME      =>  NULL,
  COMPTYPE        =>  2,
  BLKCNT_CMP      =>  o_blkcnt_cmp,
  BLKCNT_UNCMP    =>  o_blkcnt_uncmp,
  ROW_CMP         =>  o_row_cmp,
  ROW_UNCMP       =>  o_row_uncmp,
  CMP_RATIO       =>  o_cmp_ratio,
  COMPTYPE_STR    =>  o_comptype_str,
  SAMPLE_RATIO    =>  100,
  OBJTYPE         =>  1);
RAISE INFO 'Number of blocks used by the compressed sample of the object     : %', o_blkcnt_cmp;
RAISE INFO 'Number of blocks used by the uncompressed sample of the object     : %',
o_blkcnt_uncmp;
RAISE INFO 'Number of rows in a block in compressed sample of the object     : %', o_row_cmp;
RAISE INFO 'Number of rows in a block in uncompressed sample of the object     : %', o_row_uncmp;
RAISE INFO 'Estimated Compression Ratio of Sample                    : %', o_cmp_ratio;
RAISE INFO 'Compression Type                         : %', o_comptype_str;
end;
/
INFO:  Number of blocks used by the compressed sample of the object     : 0
INFO:  Number of blocks used by the uncompressed sample of the object     : 0
INFO:  Number of rows in a block in compressed sample of the object     : 0
INFO:  Number of rows in a block in uncompressed sample of the object     : 0
INFO:  Estimated Compression Ratio of Sample                    : 1
INFO:  Compression Type                         : Compress Advanced
```

6. Query the LMT of each row (index compression is not supported).

```
DBE_HEAT_MAP.ROW_HEAT_MAP(
OWNER           IN VARCHAR2,
SEGMENT_NAME    IN VARCHAR2,
PARTITION_NAME  IN VARCHAR2     DEFAULT NULL,
CTID            IN TEXT,
V_DEBUG         IN BOOL         DEFAULT FALSE);
```

Table 7-15 Input parameters of DBE_HEAT_MAP.ROW_HEAT_MAP

Name	Description
OWNER	Schema to which a data object belongs.
SEGMENT_NAME	Data object name.
PARTITION_NAME	Data object partition name. This parameter is optional. The default value is **NULL**.
CTID	**ctid** of the target row, that is, **block_id** or **row_id**.
V_DEBUG	Debugging. Debug logs are recorded.

Table 7-16 Output parameters of DBE_HEAT_MAP.ROW_HEAT_MAP

Name	Description
OWNER	Owner of a data object.
SEGMENT_NAME	Data object name.
PARTITION_NAME	Name of a data object partition. This parameter is optional.
TABLESPACE_NAME	Name of the tablespace to which data belongs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description
FILE_ID	ID of the absolute file to which a row belongs.
RELATIVE_FNO	ID of the relative file to which a row belongs. (GaussDB does not have this logic. Therefore, the value is the same as the preceding value.)
CTID	**ctid** of a row, that is, **block_id** or **row_id**.
WRITETIME	LMT of a row.

Example:

```
gaussdb=# ALTER DATABASE set ilm = on;
gaussdb=# CREATE Schema HEAT_MAP_DATA;
gaussdb=# SET current_schema=HEAT_MAP_DATA;

gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1';
gaussdb=# CREATE TABLE HEAT_MAP_DATA.heat_map_table(id INT, value TEXT) TABLESPACE
example1;
gaussdb=# INSERT INTO HEAT_MAP_DATA.heat_map_table VALUES (1, 'test_data_row_1');

gaussdb=# SELECT * from DBE_HEAT_MAP.ROW_HEAT_MAP(
    owner         =>  'heat_map_data',
    segment_name  =>  'heat_map_table',
    partition_name  => NULL,
    ctid          =>  '(0,1)');
     owner     |  segment_name  | partition_name | tablespace_name | file_id | relative_fno | ctid  |
writetime
---------------+----------------+----------------+-----------------+---------+--------------+-------+-----------
 heat_map_data | heat_map_table |                | example1        |   17291 |        17291 | (0,1) |
(1 row)
```

7. Query the environment parameters related to ILM scheduling and execution.

```
gaussdb=# SELECT * FROM GS_ADM_ILMPARAMETERS;
              name               | value
---------------------------------+-------
 EXECUTION_INTERVAL              |    15
 RETENTION_TIME                  |    30
 ENABLED                         |     1
 POLICY_TIME                     |     0
 ABS_JOBLIMIT                    |    10
 JOB_SIZELIMIT                   |  1024
 WIND_DURATION                   |   240
 BLOCK_LIMITS                    |    40
 ENABLE_META_COMPRESSION         |     0
 SAMPLE_MIN                      |    10
 SAMPLE_MAX                      |    10
 CONST_PRIO                      |    40
 CONST_THRESHOLD                 |    90
 EQVALUE_PRIO                    |    60
 EQVALUE_THRESHOLD               |    80
 ENABLE_DELTA_ENCODE_SWITCH      |     1
 LZ4_COMPRESSION_LEVEL           |     0
 ENABLE_LZ4_PARTIAL_DECOMPRESSION |     1
 INDEXJOB_SIZELIMIT              |  1024
 ENABLE_INDEX_COMPRESS           |     0
(20 rows)
```

8. Query the brief information about an ILM policy, including the policy name, type, enabling status, disabling status, and deletion status.

```
gaussdb=# SELECT * FROM GS_ADM_ILMPOLICIES;
policy_name |  policy_type  | tablespace | enabled | deleted
-------------+---------------+------------+---------+---------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 p1          | DATA MOVEMENT |            | YES     | NO


gaussdb=# SELECT * FROM GS_MY_ILMPOLICIES;
 policy_name |  policy_type  | tablespace | enabled | deleted
-------------+---------------+------------+---------+---------
 p1          | DATA MOVEMENT |            | YES     | NO
```

9. Query the brief data movement information about an ILM policy, including the policy name, action type, and condition.

```
gaussdb=# SELECT * FROM GS_ADM_ILMDATAMOVEMENTPOLICIES;
 policy_name | action_type | scope | compression_level | tier_tablespace | tier_status |
condition_type     | condition_days | custom_function | policy_subtype | action_clause | tier_to
-------------+-------------+-------+-------------------+-----------------+-------------+-------------------------
+----------------+-----------------+----------------+---------------+---------
 p1          | COMPRESSION | ROW   | ADVANCED          |                 |             | LAST MODIFICATION
TIME |             90 |                 |                |               |

gaussdb=# SELECT * FROM GS_MY_ILMDATAMOVEMENTPOLICIES;
 policy_name | action_type | scope | compression_level | tier_tablespace | tier_status |
condition_type     | condition_days | custom_function | policy_subtype | action_clause | tier_to
-------------+-------------+-------+-------------------+-----------------+-------------+-------------------------
+----------------+-----------------+----------------+---------------+---------
 p1          | COMPRESSION | ROW   | ADVANCED          |                 |             | LAST MODIFICATION
TIME |             90 |                 |                |               |
(1 row)
```

10. Query the brief information about all data objects to which ILM policies are applied and the corresponding policies, including the policy name, data object name, policy source, and policy enabling/disabling status.

```
gaussdb=# SELECT * FROM  GS_ADM_ILMOBJECTS;
 policy_name | object_owner | object_name | subobject_name | object_type |    inherited_from    |
tbs_inherited_from | enabled | deleted
-------------+--------------+-------------+----------------+-------------+----------------------
+--------------------+---------+---------
 p1          | public       | lineitem    |                | TABLE       | POLICY NOT INHERITED |                    |
YES     | NO

gaussdb=# SELECT * FROM  GS_MY_ILMOBJECTS;
 policy_name | object_owner | object_name | subobject_name | object_type |    inherited_from    |
tbs_inherited_from | enabled | deleted
-------------+--------------+-------------+----------------+-------------+----------------------
+--------------------+---------+---------
 p1          | public       | lineitem    |                | TABLE       | POLICY NOT INHERITED |                    |
YES     | NO
```

11. Query the brief information about an ADO task, including the task ID, task owner, status, and time.

```
gaussdb=# SELECT * FROM GS_ADM_ILMTASKS;
 task_id | task_owner |   state   |         creation_time         |         start_time          |
completion_time
---------+------------+-----------+-------------------------------+-----------------------------
+-------------------------------
       1 | omm        | COMPLETED | 2023-10-16 12:03:55.113296+08 | 2023-10-16 12:03:55.113296+08 |
2023-10-16 12:03:56.326864+08
(1 row)

gaussdb=# SELECT * FROM GS_MY_ILMTASKS;
 task_id | task_owner |   state   |         creation_time         |         start_time          |
completion_time
---------+------------+-----------+-------------------------------+-----------------------------
+-------------------------------
       1 | omm        | COMPLETED | 2023-10-16 12:03:55.113296+08 | 2023-10-16 12:03:55.113296+08 |
2023-10-16 12:03:56.326864+08
(1 row)
```

12. Query the evaluation details of an ADO task, including the task ID, policy information, object information, evaluation result, and ADO job name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# SELECT * FROM GS_ADM_ILMEVALUATIONDETAILS;
 task_id | policy_name | object_owner | object_name | subobject_name | object_type |
selected_for_execution |     job_name      | comments

---------+-------------+--------------+-------------+----------------+-------------+-----------------------
+--------------------+---------
N/A
       1 | p2          | public       | ilm_table_1 |                | TABLE       | SELECTED FOR EXECUTION | ilmjob
$_postgres1 |
(1 row)

gaussdb=# SELECT * FROM GS_MY_ILMEVALUATIONDETAILS;
 task_id | policy_name | object_owner | object_name | subobject_name | object_type |
selected_for_execution |     job_name      | comments

---------+-------------+--------------+-------------+----------------+-------------+-----------------------
+--------------------+---------
N/A
       1 | p2          | public       | ilm_table_1 |                | TABLE       | SELECTED FOR EXECUTION | ilmjob
$_postgres1 |
(1 row)
```

13. Query the execution details of an ADO job, including the task ID, job name, job status, and job time.

```
gaussdb=# SELECT * FROM GS_ADM_ILMRESULTS;
 task_id |     job_name      |       job_state        |          start_time           |       completion_time        |
comments |                  statistics
---------+-------------------+------------------------+-------------------------------
+-------------------------------+----------+-----------------------------------------------------
       1 | ilmjob$_postgres1 | COMPLETED SUCCESSFULLY | 2023-10-16 12:03:56.290176+08 |
2023-10-16 12:03:56.319829+08 |          |
SpaceSaving=0,BoundTime=1697429033,StartBlkNum=0,LastBlkNum=40
(1 row)

gaussdb=# SELECT * FROM GS_MY_ILMRESULTS;
 task_id | job_name | job_state | start_time | completion_time | comments | statistics
---------+----------+-----------+------------+-----------------+----------+------------
(0 rows)
 task_id |     job_name      |       job_state        |          start_time           |       completion_time        |
comments |                  statistics
---------+-------------------+------------------------+-------------------------------
+-------------------------------+----------+-----------------------------------------------------
       1 | ilmjob$_postgres1 | COMPLETED SUCCESSFULLY | 2023-10-16 12:03:56.290176+08 |
2023-10-16 12:03:56.319829+08 |          |
SpaceSaving=0,BoundTime=1697429033,StartBlkNum=0,LastBlkNum=40
(1 row)
```

7.10 Memory Management

Linux Memory Mechanism

- Physical memory

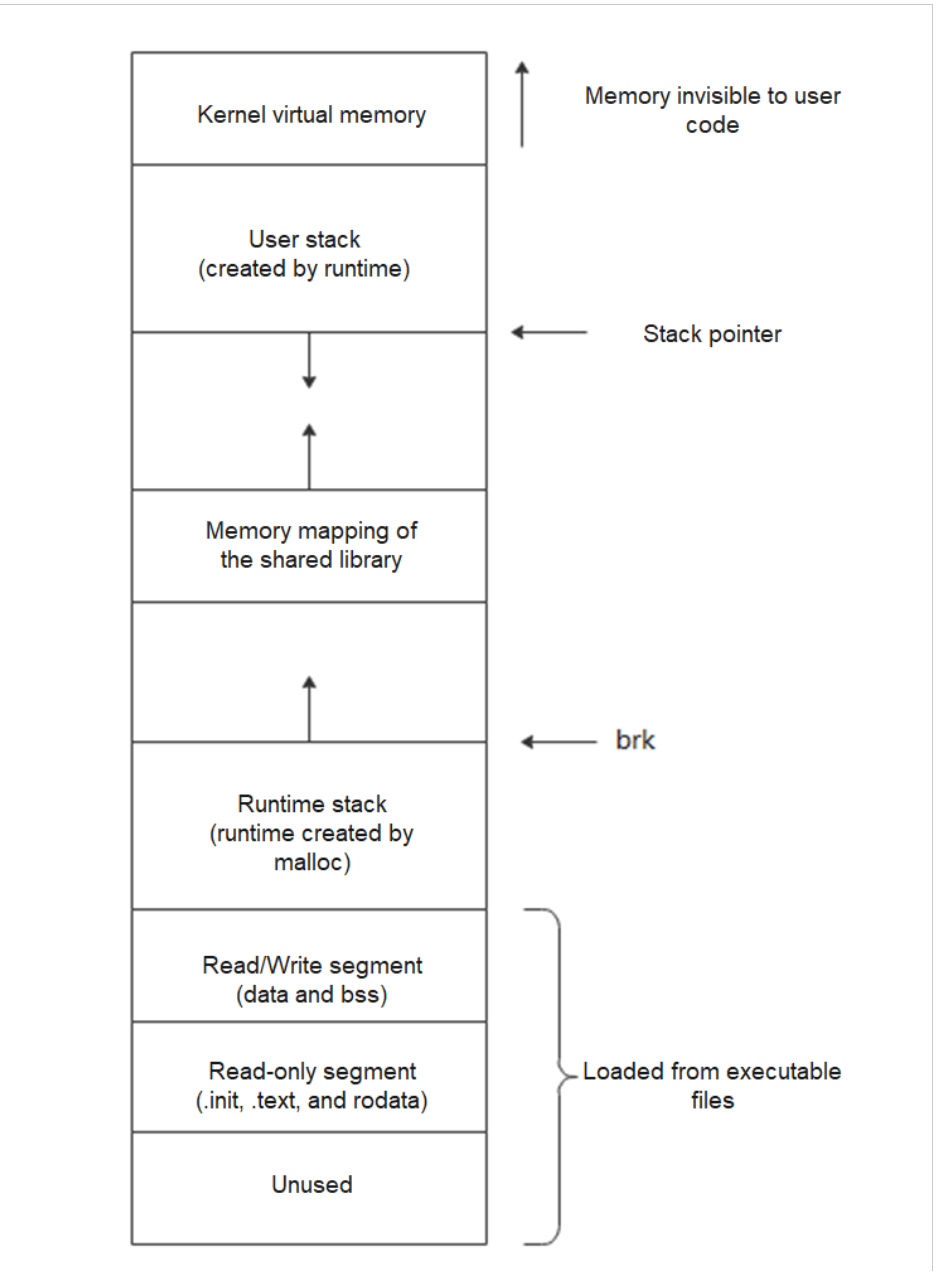
  Physically visible hardware, that is, memory modules.

- Virtual memory

  A sum of all memory spaces when a process is running. Taking a 32-bit system as an example, each process has 4 GB memory space, and the memory spaces of all the processes have similar structures, as shown in **Figure 7-19**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-19 Linux Memory Mechanism



- When a new process is created, a dedicated memory space is created. The data and code of the process are copied from the disk to this memory space when they need to be read or written. The address of each piece of data in the process is recorded by **task_struct** in the process control table. **task_struct** contains a linked list, which is used to record the allocation of memory space. The linked list records the addresses with data, the addresses without data, the readable addresses, and the writable addresses.

- The memory space allocated to each process is mapped to the corresponding disk space.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> You are advised not to limit the virtual memory of a process.

- Physical memory addresses and virtual memory addresses
  - Memory paging mechanism
    - The size of a page in the virtual memory is the same as that of a page frame in the physical memory.
    - A page table maintains a mapping from a virtual memory page to a physical memory page.
    - Page fault function: The operating system selects a virtual page frame based on a specific replacement policy, for example, least recently used (LRU), to invalidate the virtual page frame, and writes the content of the virtual page frame to the disk. Then, the page that needs to be accessed is loaded to the page frame, and the mapping relationship in the page table is updated.
  - Virtual memory address: consists of a page number (associated with a page number in a page table) and an offset (page size).
  - Virtual memory addressing: Find a frame number in the page table. If the page frame number is invalid, call the page. The page frame number and offset are sent to the memory management unit to form a physical address for accessing data in the physical memory.
- Buffer
  - Based on the disk read/write design, scattered write operations are performed in a centralized manner to reduce disk fragments and repeated disk seek, thereby improving system performance.
  - Linux has a daemon process to periodically clear the buffer (that is, write to a disk). You can also run the **sync** command to manually clear the buffer.
- Cache
  - The read data is stored. If the data is hit (the required data is found), the data is not read from the hard disk. If the data is not hit, the data is read from the hard disk. The data is organized according to the read frequency. The content that is most frequently read is placed in the location that is most easily found, and the content that is no longer read is arranged last until the content is deleted.
- OOM
  - Out of memory (OOM) means that memory cannot be recycled or too much memory is used in an application system, so that the memory required for program running is greater than maximum memory that can be allocated. In this case, the program cannot run, and the system displays a message indicating memory overflow. Sometimes, the software is automatically closed. After the computer or software is restarted, some memory is released and the software can run normally. The memory overflow error caused by system configuration, data flow, or user code cannot be avoided even if the user performs the task again.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

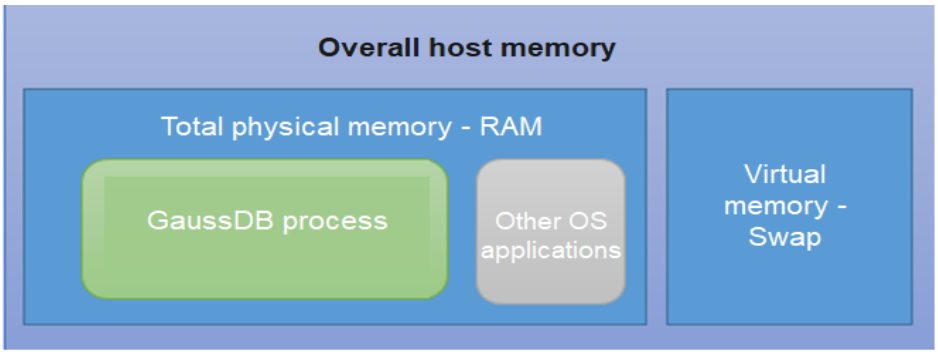
GaussDB Memory Resource Management

- Introduction to memory management
  - The dynamic memory usage of GaussDB is based on the memory context management. In terms of the memory context mechanism, the logical memory management mechanism is introduced, and multiple views are provided to trace the memory usage.

    Currently, the memory management of GaussDB is divided into two levels:

    - Node-level memory control (non-multi-primary): You can set **max_process_memory** to determine the maximum available memory of CNs or DNs.
    - Memory job-level control: You can limit the maximum memory that can be used by a single query statement.
- Node-level memory control
  - In the Linux OS, the memory space usually consists of the physical memory RAM and the virtual memory Swap. The physical memory is shared by database and non-database programs, and the memory space of database programs is shared by GaussDB processes, as shown in Figure 7-20.

    Figure 7-20 Overall host memory

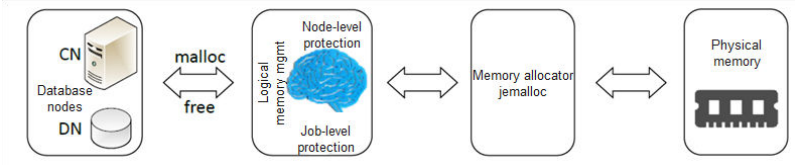
    

  - Since non-database programs also consume memory, it is generally advisable to allocate 80% of the system's available memory to GaussDB.
- Job-level memory control
  - The **query_mem** parameter specifies the amount of memory that can be used by each job based on the amount of memory that can be used and the number of jobs that can be concurrently executed. The **query_max_mem** parameter specifies the upper limit of memory that can be used by each job.
  - (Non-multi-primary) The **work_mem** parameter is provided to set the memory size used by internal sorting operations and hash tables before they are written to temporary disk files. The **maintenance_work_mem** parameter is provided to set the maximum memory that can be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY.
- Logical memory management mechanism

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

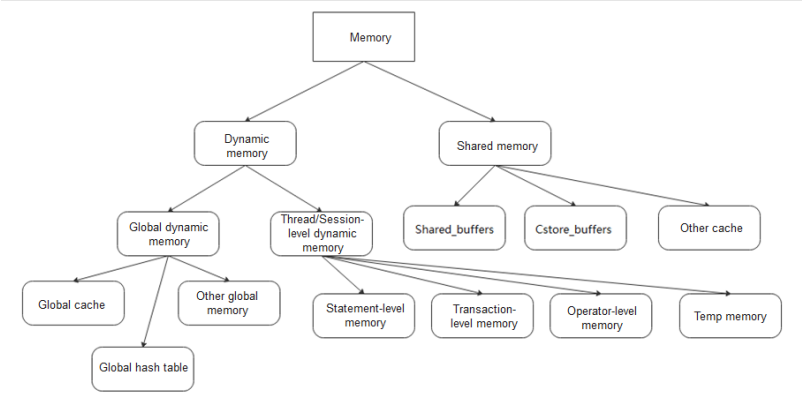
– GaussDB memory management is implemented through logical memory management. The implementation principle is to add a layer of logical memory management above the original memory allocation. The system determines whether to allocate memory to jobs by checking whether the allocated memory exceeds the specified memory. Logical memory management does not change the original memory resource allocation mechanism. A logic judgment layer is added before memory allocation to check whether the memory usage reaches the upper limit and determine whether to allocate memory, as shown in **Figure 7-21**.

Figure 7-21 Logical memory management mechanism (non-multi-primary)



- Memory types

Figure 7-22 Memory types (non-multi-primary)



The memory usage of GaussDB is classified as shown in the preceding figure. The description of each type is as follows:

– Dynamic memory

  - Global dynamic memory
    - Global cache: The global cache is used to improve access efficiency. It includes a system catalog tuple cache (SysCache) and a table schema information cache (RelCache). The SysCache stores tuples of recently used system catalogs, and RelCache contains schema information (including system catalog information) about all recently accessed tables. RelCache stores the RelationData structure instead of tuples. Each RelationData structure indicates the schema information of a table. The information is constructed from the information in the system catalog tuples.
    - Global hash table: a memory hash structure that stores global statistics or other data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

        - Other global memory: stores global variables.
      - Thread/Session-level dynamic memory
        - Statement-level memory: The memory lifecycle is the entire SQL statement execution phase. Memory is allocated before a statement is executed and is recycled and released after the statement is executed.
        - Transaction-level memory: The memory lifecycle is within the entire transaction. Before a transaction starts, memory is allocated. After the transaction ends, the memory is recycled and released.
        - Operator-level memory: The memory lifecycle is within the validity period of a single operator. After the operator is executed, the memory is released and recycled.
        - Temporary memory: memory allocated to some temporary variables, which is released in time after being used.
    - Shared cache
      - Shared_buffers

        Memory used to cache data read from disks.
      - Other caches

        Memory occupied by resources such as slots and locks used by threads.
- Memory context mechanism
  - Introduction to memory context
    - Concept: The memory context is used to control the allocation and release of memory in a unified manner. The memory context records the total size of the allocated memory, the size of each allocated memory, the file where the allocated memory is located, the row number, and the association between a memory context and other memory contexts.
    - Purpose: Based on the job execution logic in the database, the memory context management mechanism is used to improve the efficiency of query memory allocation and release.
  - Logical memory management (multi-primary)
    - The upper limit of memory usage of a single process is controlled through internal accounting of the database process.
    - You can collect statistics on the overall memory usage of processes, shared and non-shared memory contexts, and key modules (such as the communications library and pooler).
    - To activate logical memory, set **enable_memory_limit** to **on** and ensure that **max_process_memory** minus memory reserved for items such as cache exceeds 2 GB. For additional details, contact an administrator.
    - The PV_TOTAL_MEMORY_DETAIL view displays the memory usage of database processes, assisting in troubleshooting memory failures. For

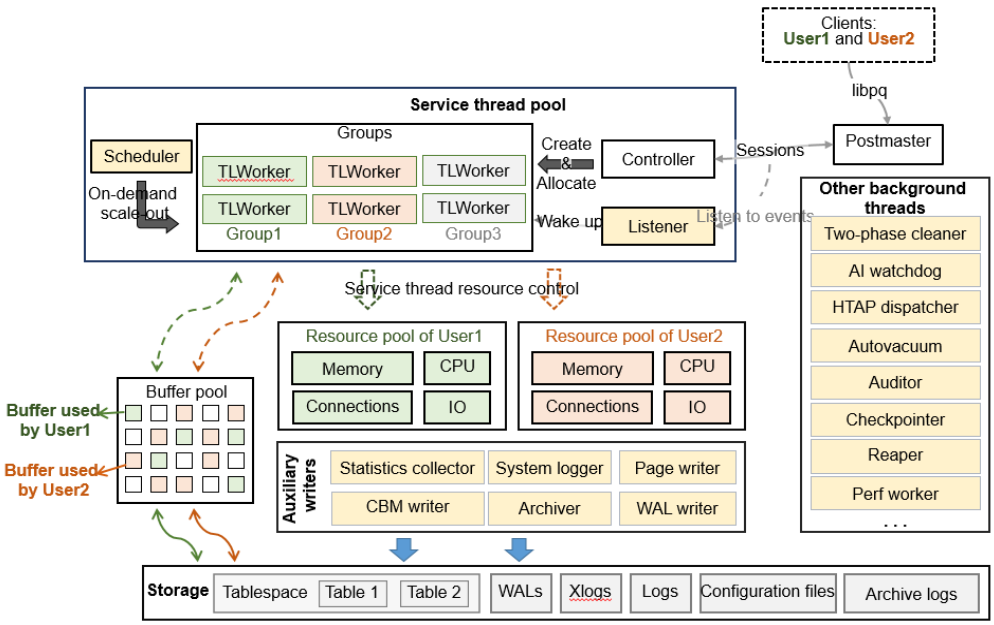
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

details about the columns available in this view, see "System Catalogs and System Views > System Views > Other System Views > PV_TOTAL_MEMORY_DETAIL" in *Reference*.

7.11 Multi-thread Architecture

GaussDB employs a single-process, multi-threaded architecture, which is a key factor contributing to its high performance and robust concurrent processing capabilities. Through effective thread scheduling and management, GaussDB achieves efficient and stable data processing in a single process. There are a diverse range of thread types and functionalities, including user threads, system threads, I/O threads, and log threads, which collaborate to accomplish various database tasks. In the multi-thread architecture, threads are easier to share resources, for example, parallel query and thread pool. **Figure 7-23** shows the GaussDB process architecture.

Figure 7-23 GaussDB process architecture



Multi-threading has the following four major advantages:

Advantage 1: Low startup overhead

The startup overhead of a thread is far less than that of a process. Compared to a process, a thread is a more frugal multitasking operation mode. In the Linux system, initiating a new process requires an independent address space and multiple data tables to preserve its code segment, stack segment, and data segment; this constitutes an "expensive" multitasking operation mode. On the contrary, multiple threads running in the same process share the same address space and most of the data. Therefore, the space required for starting a thread is significantly less than the space required for starting a process.

Advantage 2: Convenient communication mechanism

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Different processes have independent data spaces, and data transfer can only be done through communication, which is both time-consuming and inconvenient. Threads within the same process share a common data space, allowing one thread's data to be directly utilized by other threads, which not only enhances speed but also brings convenience.

Advantage 3: Low switching overhead

In the Linux system, process switching involves two steps: switching a page directory to utilize a new address space, and switching a kernel stack and a hardware context. For thread switching, it is unnecessary to switch the page directory to use the new address space. You only need to switch the kernel stack and hardware context. Therefore, the overhead of thread switching is obviously less than that of process switching.

Advantage 4: Low system call overhead

In a multi-thread architecture, system call overheads are relatively low due to minimal context switching between user and kernel modes. Process switching, however, entails additional overhead from switching between address spaces.

Main Thread Types

During GaussDB startup, the main function first starts the service daemon thread Postmaster. Subsequently, the Postmaster thread initiates other auxiliary threads; hence, the Postmaster thread serves as the parent thread of all threads. When receiving a user's SQL request, the service daemon thread initiates a worker thread, which handles the user's request, executes the SQL statement, and returns the data. In addition to the main thread, GaussDB also provides some auxiliary threads to implement maintenance and scheduling functions. **Table 7-17** describes the main threads and their functions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-17 Main thread types in GaussDB

Thread	Description
main thread	Database main thread. It provides the following functions: 1. Connection listening and receiving: • The main thread is responsible for listening to connection requests from clients and receiving new connections. • When receiving a new connection request, the main thread handles the connection and allocates necessary resources to enable the client to communicate with the database. 2. Worker thread monitoring and processing: • The main thread continuously monitors the status of all worker threads to ensure that they run properly. • It performs the following operations based on the exit status of a worker thread: – If a worker thread exits due to a FATAL error, the main thread will restart it. – If a worker thread exits due to a PANIC error, the main thread triggers the re-initialization of the entire database to ensure the stability and consistency of the database system.
worker thread	Worker thread. GaussDB will create multiple worker threads as needed to execute SQL statements. Each worker thread independently executes the SQL statement and returns the result to the main thread.
connect thread	Connection setup thread. GaussDB will create multiple connect threads as needed to process connection requests from clients. After a connection is set up, the session is handed over to a worker thread to execute SQL statements. This thread type is available only when **enable_thread_pool** and **enable_connect_thread_pool** are enabled.
startup	A thread started when the database is initialized. When the database is started, the Postmaster main thread initiates the first worker thread, which is the startup thread. This thread primarily handles redo log operations on the database to facilitate its recovery. After the redo operations are complete, if the database is not a standby node, the startup thread will exit. If the database is a standby node, the startup thread will continue to run to process newly received logs.
background writer	Background data writer thread. It periodically synchronizes the content of the database data buffer to disks to ensure data durability.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

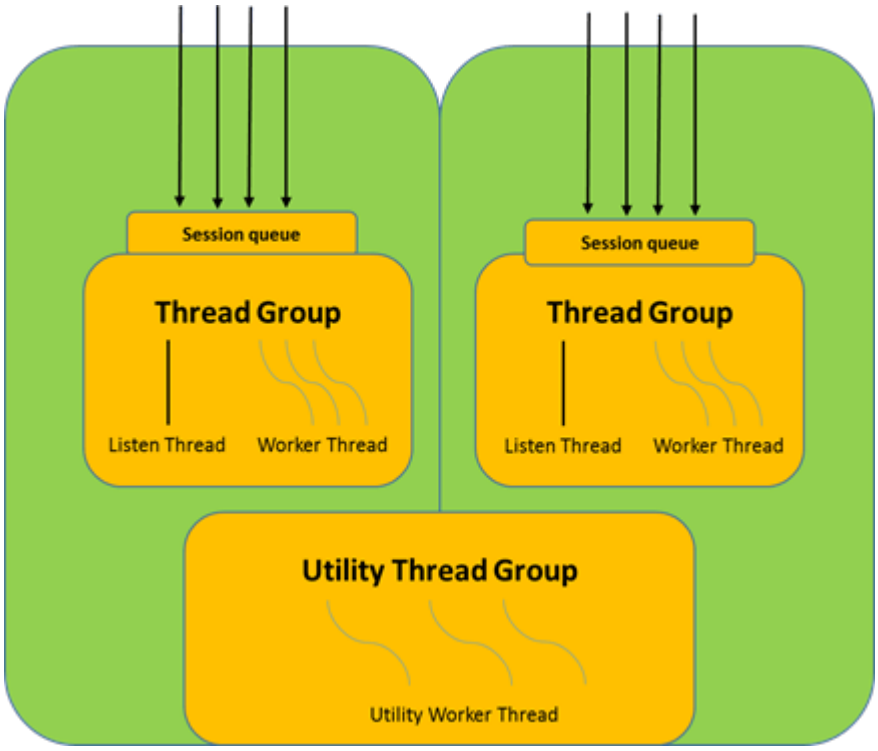
Thread	Description
checkpointer	Checkpointer thread. It performs checkpointing operations, completes periodic database checkpoints, and executes checkpoint commands to ensure database consistency and recovery capabilities.
WAL writer	Background WAL writer thread. It periodically synchronizes the contents of the log buffer to disks to ensure the durability of transaction logs.
system logger	Run log writer thread. It writes the run logs of each thread into the run log file for system monitoring and troubleshooting purposes.
autovacuum launcher	Cleaner startup thread. Its primary function is to notify the Postmaster to initiate cleaner threads and distribute the workload among these threads for enhanced system efficiency.
autovacuum worker	Cleaner thread. It cleans junk data from GaussDB to help release storage space and enhance system performance.
archiver	Log archive thread. It completes the archiving process by copying online logs to the archive directory to ensure long-term storage and backup of logs.

Thread Pool Mechanism

In a high-concurrency system environment, system resource contention (including external resources such as CPU, network, and I/O, as well as internal resources like critical lock structures) often leads to significant deterioration of system performance and a decline in system stability. Therefore, the thread pool technology is employed to decouple sessions from background worker threads, ensuring that resources remain under control and all services are executed in an orderly fashion, thereby avoiding resource contention. **Figure 7-24** shows the GaussDB thread pool model.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-24 GaussDB thread pool model



The overall design idea of the thread pool technology is to pool thread resources and directly reuse them in different connections. After the system starts up, it initiates a specific number of worker threads based on the current user configuration. A worker thread can serve one or more connection sessions. When receiving a new request, the thread pool allocates an idle thread to process the request. After the processing is complete, the thread returns to the thread pool and waits for the next request. This approach avoids frequent thread creation and destruction, thus enhancing system responsiveness and resource utilization.

GaussDB's thread model adopts the following design to effectively handle concurrent requests and enhance the database's performance and responsiveness.

- Main thread: manages connections and schedules requests.
- Worker thread pool: processes client requests.
- Concurrency control mechanism: ensures data consistency and concurrency.

7.12 ABO

The AI-based optimizer (ABO) enhances the accuracy of the optimizer in cardinality estimation, cost estimation, and plan selection using AI algorithms, thus preventing slow SQL statements and improving the overall performance of the database. Currently, intelligent (multi-column) cardinality estimation, adaptive plan selection, and adaptive cost estimation are supported.

7.12.1 Cardinality Estimation

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.12.1.1 Overview

Description

Cardinality estimation (CE) uses the Bayesian network model in the library to model the joint distribution of multi-column data samples. This process enables it to provide more accurate CE results for multi-column equality queries. On this basis, the accuracy of the optimizer in selecting plans and operators can be significantly improved. Finally, the improvement of the optimizer accuracy will enhance of the overall throughput of the database.

Scenarios

In high-load service environments such as core systems of large enterprises and cloud databases, complex queries (especially those involving multi-column combination conditions) often fail to select the expected indexes due to inaccurate cost estimation of the optimizer, resulting in a large number of slow SQL statements. The root cause of this problem is as follows: When there is significant data skew and strong correlation between multiple columns, the cardinality estimator that relies on traditional statistics (such as histograms) or independence assumptions cannot accurately predict the size of the query result set. As a result, the execution plan is suboptimal. These slow SQL statements not only significantly increase the response time of key service APIs, but also exhaust database resources, severely affecting the service success rate and overall system availability. This functionality aims to describe multidimensional data distribution and provide the optimizer with accurate selectivity estimation capabilities by using a lightweight data statistics model, thereby improving the cardinality estimation accuracy in multi-column scenarios and end-to-end execution performance.

Specifications and Limitations

This feature is available since 503.1.0. After this feature is used, the query performance may be affected by the number of cached models. If the number of accessed models exceeds that of currently cached models, memory replacement and disk access may occur, which may cause performance deterioration. Therefore, you are advised to set the number of cached models properly. The number of models is equal to that of created Bayesian network statistics columns. You are advised to set the number of cached models to that of statistics columns that can be accessed within a period. Currently, the default number of supported cached models is 100. The actual number can be adjusted using **ai_stats_cache_limit**.

Technical Architecture

The intelligent cardinality estimation solution integrates the lightweight probabilistic graphical model into the traditional statistics module of GaussDB. The model is trained during statistics collection and stored in the system catalogs for the optimizer to use.

The database receives specific analytical query statements, determines whether the statements contain multiple columns, and calls the data statistics module. After receiving the ANALYZE instruction, the data statistics module samples data from the columns and collects statistics on the data samples. If the data statistics module detects that the intelligent statistics GUC parameter is enabled, the data statistics module creates a Bayesian network model, including:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Use the aggregation operation to collect statistics and calculate the correlation between every two columns.
- Use the Chow-Liu algorithm to generate a tree Bayesian network structure.
- Call the Bayesian network operator to traverse data samples for model parameter training, save the model parameters in binary mode to the system model table gs_model_warehouse, and return model metadata such as the model name.
- Write the model metadata to the system statistics table.
- If multi_stats_type is set to **"ALL"**, traditional statistics are also stored in the model.

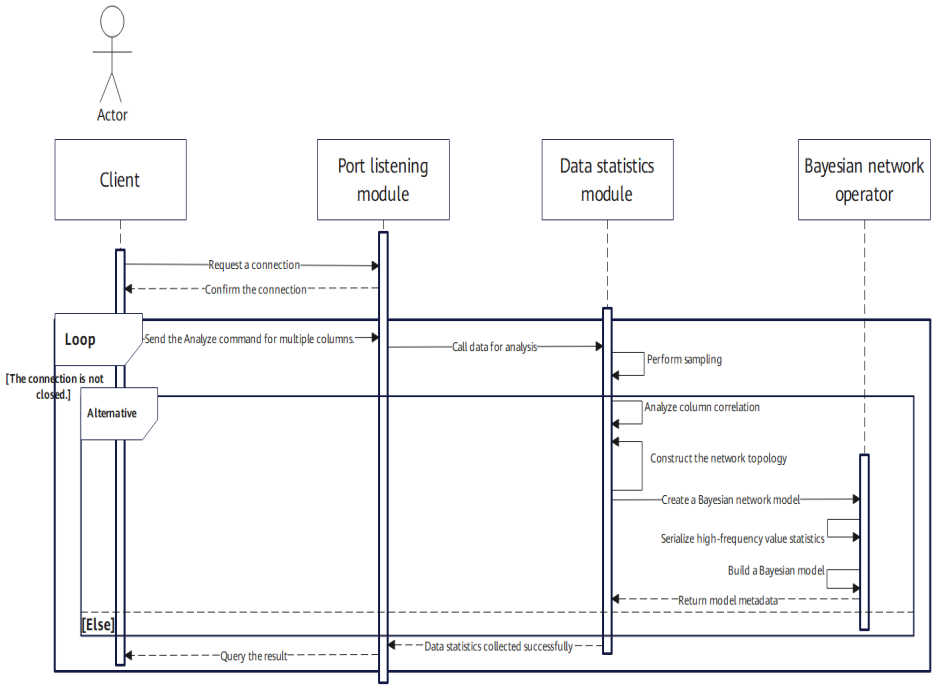
7.12.1.2 Technical Principles

Core Mechanism

- Model creation

  This functionality is triggered by the **ANALYZE** command or auto-analyze committed by users. The CN sample collector obtains the data sample structure in the memory and packs it into a data source structure. As shown in **Figure 7-25**, the data source structure contains two parts: high-frequency value statistics and full data sample information. High-frequency value statistics include values whose frequency is greater than 1 (excluding values whose frequency is 1). Full sample information is flattened to a sequence and sent to a model one by one. Therefore, the model can determine whether the data read currently is high-frequency value or common sample data based on the obtained frequency. High-frequency value statistics are directly serialized and saved to the model by its operators. For common sample data, the model operators performs Bayesian network model statistics collection on the part without high-frequency values.

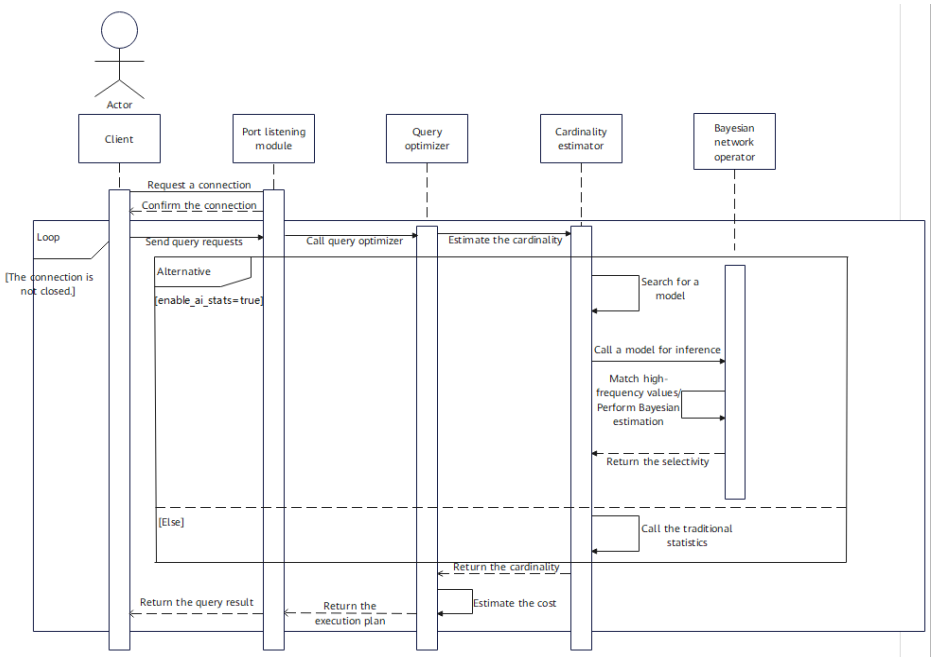
  Figure 7-25 Model creation sequence



- Model inference

  In the cardinality estimation phase, the system reads the model from the disk to the memory, deserializes the model to the memory structure, and uses the independent memory contexts for caching. As shown in **Figure 7-26**, for each multi-column equality query, the system first searches the high-frequency value hash table. If the high-frequency value hash table is found, the system directly returns the selectivity P1(t). Otherwise, the system calculates the selectivity P2(t) of the query condition in the Bayesian network model, and finally returns P2(t) / (1.0-SUM(P1)), where SUM(P1) is the sum of all high-frequency value selectivities. After this functionality is used, the priority of multi-column equality selectivity estimation in GaussDB is as follows: If high-frequency values exist, the system preferentially obtains them from the high-frequency value hash table. If non-high-frequency values exist, the system obtains the selectivity through the Bayesian network. If the preceding models are not created or overwritten, the system attempts to use multi-column NDV for estimation. If the statistics do not match the queried columns, the system uses multi-column function dependency. If none of the preceding statistics is created, the clause condition is added back to the unmatched condition, and then the "single column+independence assumption" method is used for estimation.

  Figure 7-26 Model inference sequence



- Model information display

  Call the system function gs_ai_stats_explain(relid, stakeys). The value of **relid** parameter is an integer, and that of the **stakeys** parameter is an array. After obtaining the input parameters, the system function matches them with the statistics row of the BAYESNET type in the system catalog pg_statistic_ext. If the row cannot be found, a failure message is displayed. Otherwise, the system calls the explain function of the Bayesian network, a model explanation framework function, to deserialize the model read from the disk, recombine the statistics into an array of a readable type, and return the array to the system function. The main content includes the Bayesian network

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

topology, high-frequency values and their selectivities, bucket types on a single column, histogram or MCV boundary values, and probability of each possible value in each bucket. Other information includes common model information, such as the model training time, model hyperparameters, and model creation time.

Performance Metrics

- The accuracy of multi-column equality query on highly skewed correlated data and the end-to-end performance in typical scenarios involving index selection are effectively improved.
- No deterioration occurs in scenarios such as TPC-C.

API Description

Built-in system functions

Function	Description
gs_ai_stats_explain	Prints the intelligent statistics of multiple columns in the corresponding table and columns.

7.12.1.3 Operation Guide

Prerequisites

- Software version: kernel version ≥ GaussDB Kernel 503.1.0.B038
- The database is working properly.
- The GUC parameter **enable_ai_stats** is set to **on**.
- **multi_stats_type** is set to **'BAYESNET'** or **'ALL'**.
- The GUC parameter **enable_fast_query_shipping** is set to **off**.

Procedure

Step 1 Sample by sampling rate. That is, set the GUC parameter **default_statistics_target** to an integer ranging from –100 to –1, indicating the sampling percentage.

Step 2 Use **ANALYZE table_name((column1_name,column2_name [, ...]))** to collect statistics and create a model.

Step 3 Enter a query. If a statistical model is created on the equality query column involved in the query, the statistical model is automatically used to estimate the selectivity.

Step 4 If the intelligent statistics model is no longer required, run **ALTER TABLE table_name DELETE STATISTICS ((column1_name,column2_name[, ...]))** to delete the statistics and model.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about other usage methods, see "SQL Reference > SQL Syntax > A > ALTER TABLE" and "SQL Reference > SQL Syntax > A > ANALYZE | ANALYSE" in *Reference*.

Examples

The execution plan in the example may be different from the actual one due to differences in the environment, version, and deployment mode. This example is for reference only. The actual optimization effect is subject to the actual execution plan.

```
-- Create a table and insert data into the table.
gaussdb=# DROP TABLE IF EXISTS t1;
DROP TABLE
gaussdb=# CREATE TABLE t1(a INT , b INT, c INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(GENERATE_SERIES(1, 1000),1,1);
INSERT 0 1000
-- Check GUC parameters.
gaussdb=# SHOW enable_ai_stats;
 enable_ai_stats
-----------------
 on
(1 row)
gaussdb=# SHOW multi_stats_type;
 multi_stats_type
------------------
 bayesnet
(1 row)
gaussdb=# SHOW enable_fast_query_shipping;
 enable_fast_query_shipping
----------------------------
 off
(1 row)
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
-- Create a statistical model.
gaussdb=# ANALYZE t1((a,b));
ANALYZE
-- Query the statistical model.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT gs_ai_stats_explain('t1', ARRAY['a', 'b']);
-[ RECORD 1 ]-------
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
--------
gs_ai_stats_explain | Name: BAYESNET_MODEL_STATS_xxx_xxxxx
                    | Algorithm: bayes_net_internal
                    | Query: DIRECT ML API
                    | Return type: Float64
                    | Pre-processing time: 0.000000
                    | Execution time: 0.003121
                    | Processed tuples: 1000
                    | Discarded tuples: 0
                    | edges_of_network: 0,1
                    | num_edges: 1
                    | num_nodes: 2
                    | num_bins: 500
                    | num_mcvs: 500
                    | num_sample_rows: 1000
                    | num_total_rows: 1000
                    | Number of nodes: 2
                    | Bucket Type: Range Bin
                    | Column #0:
{1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,
77,79,81,83,85,87,89,91,93,95,97,99,101,103,105,107,109,111,113,115,117,119,121,123,125,127,129,131,133,1
35,137,139,141,143,145,147,149,151,153,155,157,159,161,163,165,167,169,171,173,175,177,179,181,183,185,
187,189,191,193,195,197,199,201,203,205,207,209,211,213,215,217,219,221,223,225,227,229,231,233,235,237
,239,241,243,245,247,249,251,253,255,257,259,261,263,265,267,269,271,273,275,277,279,281,283,285,287,28
9,291,293,295,297,299,301,303,305,307,309,311,313,315,317,319,321,323,325,327,329,331,333,335,337,339,3
41,343,345,347,349,351,353,355,357,359,361,363,365,367,369,371,373,375,377,379,381,383,385,387,389,391,
393,395,397,399,401,403,405,407,409,411,413,415,417,419,421,423,425,427,429,431,433,435,437,439,441,443
,445,447,449,451,453,455,457,459,461,463,465,467,469,471,473,475,477,479,481,483,485,487,489,491,493,49
5,497,499,501,503,505,507,509,511,513,515,517,519,521,523,525,527,529,531,533,535,537,539,541,543,545,5
47,549,551,553,555,557,559,561,563,565,567,569,571,573,575,577,579,581,583,585,587,589,591,593,595,597,
599,601,603,605,607,609,611,613,615,617,619,621,623,625,627,629,631,633,635,637,639,641,643,645,647,649
,651,653,655,657,659,661,663,665,667,669,671,673,675,677,679,681,683,685,687,689,691,693,695,697,699,70
1,703,705,707,709,711,713,715,717,719,721,723,725,727,729,731,733,735,737,739,741,743,745,747,749,751,7
53,755,757,759,761,763,765,767,769,771,773,775,777,779,781,783,785,787,789,791,793,795,797,799,801,803,
805,807,809,811,813,815,817,819,821,823,825,827,829,831,833,835,837,839,841,843,845,847,849,851,853,855
,857,859,861,863,865,867,869,871,873,875,877,879,881,883,885,887,889,891,893,895,897,899,901,903,905,90
7,909,911,913,915,917,919,921,923,925,927,929,931,933,935,937,939,941,943,945,947,949,951,953,955,957,9
59,961,963,965,967,969,971,973,975,977,979,981,983,985,987,989,991,993,995,997,999,1000}
                    | Column Probabilities:
{.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001
,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,
.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.
001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,.001,0}
                    | Bucket Type: All Values
                    | Column #1: {1}
                    | Column Probabilities: {1}
                    |
gaussdb=# \x
Expanded display is off.
```

```
-- The query is successful.
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t1 WHERE a=1 AND b=1;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..12.54 rows=1 width=12) (actual time=4.944..5.275 rows=1 loops=1)
   Node/s: datanode1
   ->  Seq Scan on t1  (cost=0.00..12.45 rows=1 width=12) (actual time=[0.226,0.576]..[0.226,0.576], rows=1)
         Filter: ((a = 1) AND (b = 1))
         Rows Removed by Filter: 491
 Total runtime: 5.785 ms
(6 rows)
-- If the intelligent statistics model is not required, delete it.
gaussdb=# ALTER TABLE t1 DELETE STATISTICS((a,b));
ALTER TABLE
-- Drop a table.
gaussdb=# DROP TABLE t1;
DROP TABLE
```

7.12.1.4 Best Practices

The following table and data come from TPC-H 50X.

Generate the following data table:

```
gaussdb=# \d part;
id             | integer                  |
p_brand        | character varying(256)   |
p_type         | character varying(256)   |
p_container    | character varying(256)   |
p_mfgr         | character varying(256)   |
```

Insert 10,000,000 lines of data.

```
gaussdb=# SELECT COUNT(1) FROM part;
10000000
```

Create four different multi-column indexes on the data table.

```
gaussdb=# SELECT * FROM pg_indexes WHERE tablename='part';
public     | part     | brand_type_container  |             | CREATE INDEX brand_type_container ON part USING
btree (p_brand, p_type, p_container) TABLESPACE pg_default
public     | part     | brand_type_mfgr       |             | CREATE INDEX brand_type_mfgr ON part USING btree
(p_brand, p_type, p_mfgr) TABLESPACE pg_default
public     | part     | brand_container_mfgr  |             | CREATE INDEX brand_container_mfgr ON part USING
btree (p_brand, p_container, p_mfgr) TABLESPACE pg_default
public     | part     | type_container_mfgr   |             | CREATE INDEX type_container_mfgr ON part USING
btree (p_type, p_container, p_mfgr) TABLESPACE pg_default
```

Generate a batch of queries that contain multi-column equal conditions for the data table.

```
gaussdb=# EXPLAIN ANALYZE SELECT * FROM part WHERE p_container='LG CASE' AND
p_brand='Brand#34' AND p_mfgr='Manufacturer#2' AND p_type='SMALL BRUSHED COPPER';
```

Test the execution plan in the scenarios where multi-column statistics are not created and ABO statistics are created.

```
gaussdb=# EXPLAIN ANALYZE SELECT * FROM part WHERE p_container='LG CASE' AND
p_brand='Brand#34' AND p_mfgr='Manufacturer#2' AND p_type='SMALL BRUSHED COPPER';
Bitmap Heap Scan on part (cost=5.30..336.06 rows=17 width=56) (actual time=0.953..7.061 rows=103
loops=1)
  Recheck Cond: (((p_brand)::text = 'Brand#34'::text) AND ((p_type)::text = 'SMALL BRUSHED COPPER'::text)
AND ((p_container)::text = 'LG CASE'::text))
  Filter: ((p_mfgr)::text = 'Manufacturer#2'::text)
  Rows Removed by Filter: 773
  Heap Blocks: exact=871
  ->  Bitmap Index Scan on brand_type_container  (cost=0.00..5.30 rows=84 width=0) (actual
time=0.704..0.704 rows=876 loops=1)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    Index Cond: (((p_brand)::text = 'Brand#34'::text) AND ((p_type)::text = 'SMALL BRUSHED COPPER'::text)
AND ((p_container)::text = 'LG CASE'::text))
Total runtime: 7.213 ms
gaussdb=# EXPLAIN ANALYZE SELECT * FROM part WHERE p_container='LG CASE' AND
p_brand='Brand#34' AND p_mfgr='Manufacturer#2' AND p_type='SMALL BRUSHED COPPER';
Bitmap Heap Scan on part  (cost=10.59..723.97 rows=210 width=56) (actual time=0.112..0.434 rows=103
loops=1)
  Recheck Cond: (((p_type)::text = 'SMALL BRUSHED COPPER'::text) AND ((p_container)::text = 'LG
CASE'::text) AND ((p_mfgr)::text = 'Manufacturer#2'::text))
  Filter: ((p_brand)::text = 'Brand#34'::text)
  Rows Removed by Filter: 64
  Heap Blocks: exact=167
  ->  Bitmap Index Scan on type_container_mfgr  (cost=0.00..10.54 rows=183 width=0) (actual
time=0.081..0.081 rows=167 loops=1)
    Index Cond: (((p_type)::text = 'SMALL BRUSHED COPPER'::text) AND ((p_container)::text = 'LG
CASE'::text) AND ((p_mfgr)::text = 'Manufacturer#2'::text))
Total runtime: 0.533 ms
```

Through the aforementioned operations, it can be observed that ABO cardinality estimation improves the query speed.

The execution plan in the example may be different from the actual one due to differences in the environment, version, and deployment mode. This example is for reference only. The actual optimization effect is subject to the actual execution plan.

7.12.1.5 Typical Issues

- Issue: An error is reported during training.
  ```
  Model training failed, error XXX.
  ```
  Solution:

  Check the training object based on the error information and train again.
- Issue: An error is reported for gs_ai_stats_explain.
  ```
  Cannot find table XXX.
  ```
  Solution:

  Check whether the input parameter is a correct table name.
- Issue: An error is reported for gs_ai_stats_explain.
  ```
  Column XXX of relation XXX does not exist.
  ```
  Solution:

  Check whether the input parameter is a correct column name.
- Issue: An error is reported for gs_ai_stats_explain.
  ```
  The AI statistic does not exist.
  ```
  Solution:

  Check whether the input parameter is a correct table or column name, or whether the corresponding Bayesian model has been created.

7.12.2 Adaptive Cost Estimation

7.12.2.1 Overview

Description

Adaptive cost estimation is an enhanced feature of the intelligent optimizer introduced in 505.1.0. It consists of two modules: cardinality estimation feedback

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

and adaptive cost parameter adjustment. This feature continuously collects the actual execution results of historical queries (the actual cardinality, or number of rows and execution time at the operator level in the query plan), automatically trains and updates the cardinality estimation feedback model based on the data, and dynamically calibrates the cost model parameters based on the actual execution results. The cardinality estimation feedback model can establish a mapping between query conditions and actual cardinality. The trained model can be used in real time for new queries to predict the cardinality of operators. In addition, the model can be used together with the adjusted cost model parameters to significantly improve the accuracy of optimizer cost estimation in complex scenarios and generate better execution plans.

Scenarios

In high-load service environments such as core systems of large enterprises and cloud databases, suboptimal query plans are often generated for complex queries due to inaccurate optimizer cost estimation, resulting in a large number of slow SQL statements. The root cause of this problem is that the cardinality estimator cannot accurately predict the size of the query result set, and the cost model parameters of the optimizer may not be configured appropriately. The combination of these two factors leads to the generation of suboptimal execution plans. These slow SQL statements not only significantly increase the response time of key service APIs, but also exhaust database resources, severely affecting the service success rate and overall system availability.

Specifications and Limitations

- Only the cardinality estimation of base table scan operators (SeqScan, IndexScan, IndexOnlyScan, BitmapHeapScan, and BitmapIndexScan) and join operators (NestLoop, MergeJoin, and HashJoin) is supported.
- Only the following query conditions are supported: simple join conditions (col1::type =/<>/>/< col2::type, with type conversion supported), numeric range conditions (col1::type =/<>/>/< const, where BETWEEN is processed as greater than and less than conditions, and const supports OID, numeric, int, float, and timestamp), simple string comparison conditions (col1::type =/<>/ LIKE/NOT LIKE const, where const supports BPCHAR, VARCHAR, and TEXT), and IN/OR conditions.
- Estimation cannot be performed on partitioned tables and materialized views.
- The execution plan may deteriorate along with collection of new SQL statements. Only the optimizer that has collected sufficient SQL statement information can gradually converge to generate the optimal plan.
- After feedback cost calibration, the intelligently calibrated cost parameters apply only to cost estimation of the SeqScan, IndexScan, IndexOnlyScan, BitmapHeapScan, HashJoin, NestLoop, and MergeJoin operators. When the optimizer estimates costs, if the type and condition of an operator and its suboperators are within the range supported by the cardinality estimation feedback, the cardinality and calibrated cost of the feedback model are used for cost estimation. Otherwise, the default method is used.

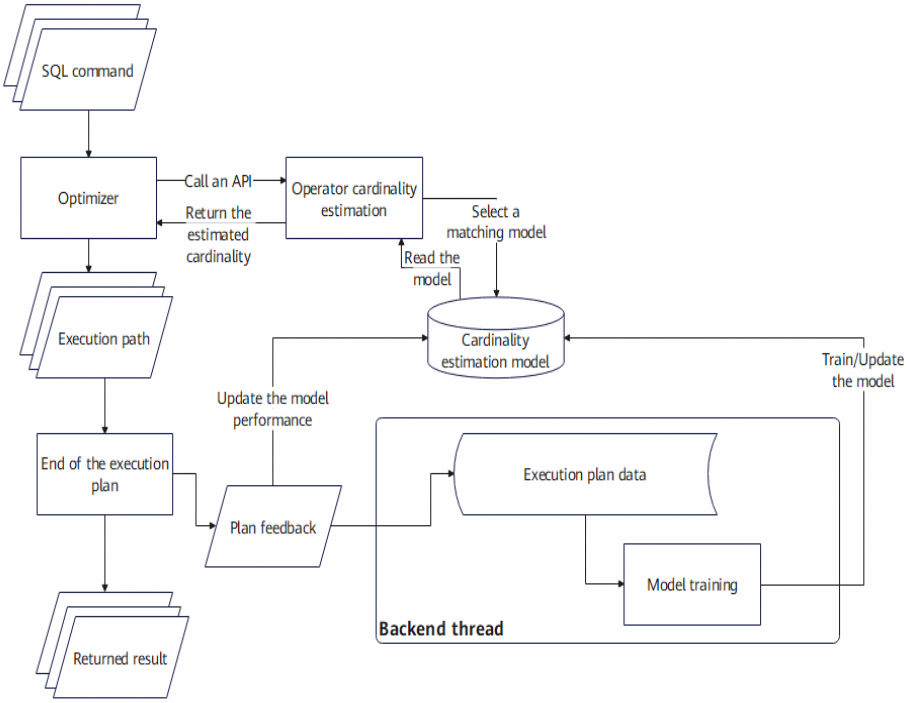
Technical Architecture

- The cardinality estimation feedback module works with the optimizer and executor in two phases: training and estimation, as shown in **Figure 7-27**. In

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the training phase, the module collects the plan feedback information (mainly the actual cardinality of each operator in the execution plan tree) of historical queries from the executor in real time, uses the data to train and iteratively update the prediction cardinality estimation feedback model, and persists the optimized model to the system catalog gs_model_warehouse. In the estimation phase, when the optimizer performs query optimization, it calls the prediction API of the module. The API automatically loads the corresponding model from gs_model_warehouse, predicts the cardinality of operators in the current query plan in real time, and sends the high-precision prediction result back to the optimizer. Model training is performed asynchronously by a background thread. When the number of collected operator samples reaches the preset threshold or the prediction error of the default cardinality estimation model exceeds a specific threshold, the thread is woken up. Users can also manually call a system function to trigger the update task.

Figure 7-27 Architecture of cardinality estimation feedback

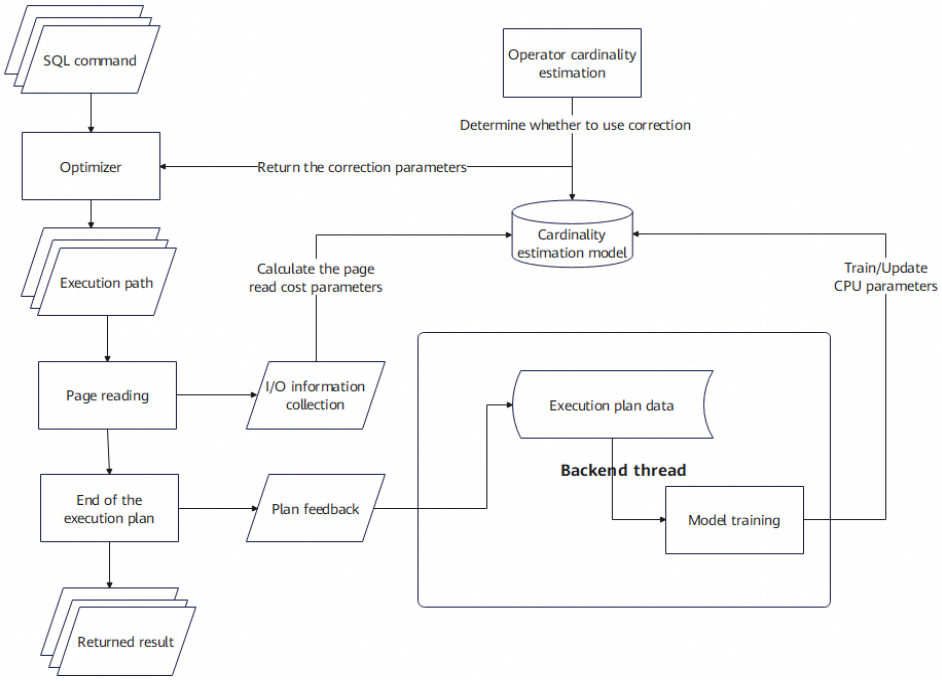


- The training information for adaptive cost parameter adjustment is also collected in collaboration with the executor, which mainly includes two types of data: I/O time consumption information and execution plan information. For I/O time consumption, the system accurately measures and accumulates the time consumed by a single I/O operation and the total number of read pages during the actual data page read when specific predefined conditions are met (for example, the module initialization is complete and the number of collected data pages does not reach the upper limit), as shown in **Figure 7-28**. The total amount of the collected information is limited to ensure that the impact on system performance is negligible. For execution plan information, the collection occurs when the query execution ends, provided that the cost parameters have not been trained. In this case, the system checks whether valid data is read during the query. If no data is read, the system collects runtime statistics of key operators (such as the Scan operator)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

in the execution plan tree, including the number of loops, number of processed tuples, CPU overhead of related computing operations, and execution duration. The collection of the two types of information is low-cost and controlled, providing necessary input for continuously optimizing the core cost model.

Figure 7-28 Framework of adaptive cost parameter adjustment



7.12.2.2 Technical Principle

Core Mechanism

- Cardinality estimation feedback

  Feedback cardinality estimation first dynamically matches a prediction model for each operator to be predicted. The matching logic is based on the unique hash code of the operator feature, which integrates information such as the tables, attributes, and query conditions involved in the operator. In addition, another mechanism is to use operator-level historical data to train and apply the unified mixture model (UMM) and K-nearest neighbors (KNN) model to model the relationship between query conditions and actual cardinalities.

  a. Calculation of the operator hash value:

     To accurately match models, the system generates a unique feature identifier (hash value) for each query operator. The core of the identifier is to capture key pattern features of operator query conditions. A query operator is described by four basic elements:

     - Attribute group (equivalence class): a set of attributes with equivalence relationships (for example, {a.id, b.id, c.id} can be extracted from the join condition a.id = b.id AND b.id = c.id).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Comparison operator: such as =, >, and < in the query condition.
- Condition value: a specific value (for example, a.id = 5) or another attribute group (for example, a.id < b.id).
- Condition type: type of a query condition, including filter query conditions (for example, a.id = 5) and join query conditions (for example, a.id = b.id).

When calculating the hash value of a given operator, the system extracts the preceding four elements, calculates the hash value of the four elements, and then calculates the total hash value. In particular, the hash value of the equivalence class is generated by combining the hash values of all attributes contained in the equivalence class.

b. UMM building:

The parameters of the UMM:

- Boundary value of the *n*-dimensional query condition space, which defines the coverage of the model.
- *M* hypercubes obtained through sampling. These hypercubes are located in the *n*-dimensional query condition space.
- Weight corresponding to each sampled hypercube.

For a query operator to be predicted, the UMM cardinality estimation process is as follows:

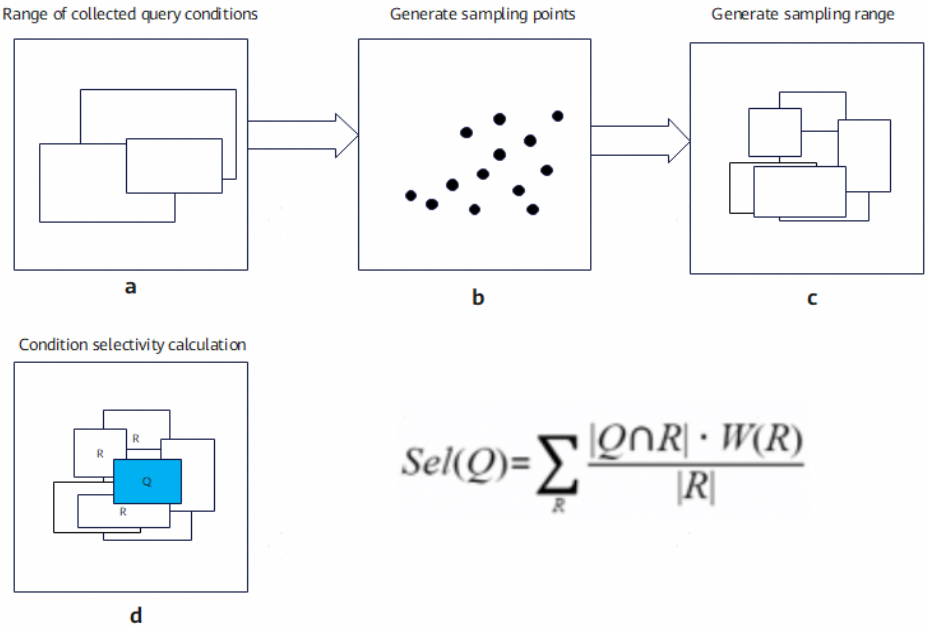
- Determine a queried hypercube corresponding to the operator in the *n*-dimensional query condition space based on the query condition of the operator.
- Calculate the *n*-dimensional measure of overlap (the area in the two-dimensional case, the volume in the three-dimensional case, and so on) between the queried hypercube and each sampled hypercube.
- Multiply the measure of each overlap by the weight of the corresponding sampled hypercube.
- Sum up all the weighted measures. The sum is the predicted cardinality of the query operator.

The following uses the two-dimensional case (*n*=2) in **Figure 7-29** as an example:

- The query condition space is a two-dimensional plane, and the sampled hypercube is a rectangle (that is, a two-dimensional hypercube) on the plane. The UMM assumes that data in each sampled rectangle is distributed in a unified manner.
- For the operator to be predicted, the query condition corresponds to a rectangular area (blue rectangle in the figure).
- The prediction cardinality of the queried rectangle is equal to the weighted sum (using the weight of the corresponding sampled rectangle) of the areas of overlaps between the queried rectangle and all sampled rectangles.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-29 UMM example



The training process of the UMM is about building a sampling hypercube and determining its weight. Specifically, based on the collected query condition information of historical query operators, an initial hypercube set (representing the entire or partial query space) covering these conditions is determined, and *M* hypercubes (subintervals) are sampled from these initial hypercubes. Then, the gradient descent algorithm is used to train and optimize the weight parameters of the *M* sampled hypercubes. The optimization objective is to minimize the cardinality prediction error of the model on the training data.

c. KNN model building:

The core idea of the KNN model is to use historical query data to establish a mapping relationship between the selectivity feature of the query condition and the final query cardinality.

Assume that an operator type contains *N* query conditions. Each query condition has a default estimated selectivity. These selectivity values can be combined into a feature vector (usually denoted as *x*). The actual query cardinality corresponding to the operator is the target value (usually denoted as *y*).

The KNN model itself does not store complex mathematical parameters. Its core "knowledge" comes from the historical query samples stored in it. Specifically, a KNN model for a specific query pattern stores a maximum of *K* historical samples. Each sample records the feature vector *x* of a similar query in the past (that is, the default estimated selectivity value of each query condition at that time) and the corresponding actual query cardinality *y*.

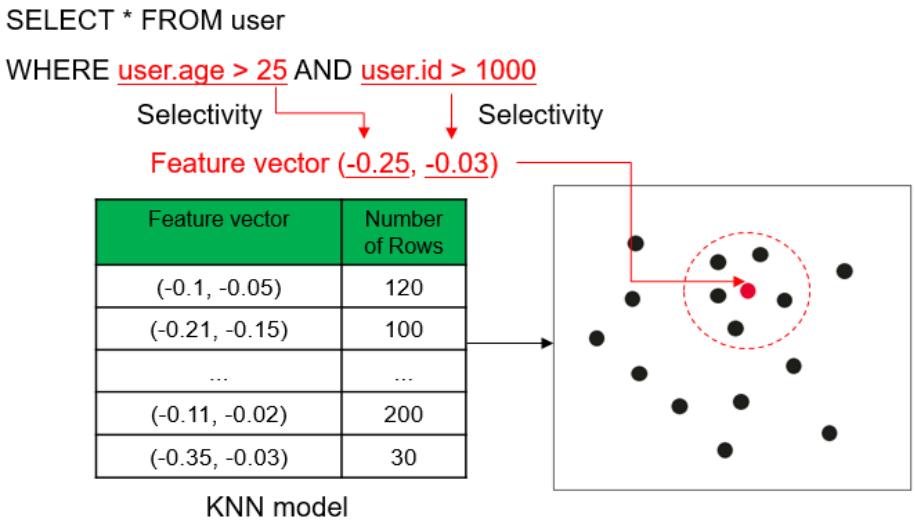
During prediction, the query optimizer first finds the corresponding KNN model based on the features of the query operator to be estimated (usually through hash matching). Then, the optimizer extracts the default estimated selectivity value of each condition in the current query to form the current feature vector *x*. The model then finds the *K* nearest neighbor

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

samples that are most similar to the current feature vector *x* from the *K* historical samples stored in the model. The final predicted query cardinality value ($\hat{y}$) is obtained by performing weighted averaging on the actual cardinality (y) of the *K* neighbor samples. The weight in the weighted averaging depends on the similarity between the feature vector (x) of each neighbor sample and the current query feature vector (x). The more similar the two vectors are (the nearer), the greater the weight of the actual cardinality in the prediction. The model uses a specific similarity measurement function (based on the Euclidean distance between vectors) to calculate this weight.

As shown in **Figure 7-30**, in a scenario in which a query condition is 2, each feature vector can be represented by a point on a two-dimensional plane, and the black points on the right side of the figure represent training data in the KNN model. In a prediction process, based on a selectivity of a to-be-predicted query condition, a feature vector of the query condition and its location (the red point in the figure) can be obtained, and then according to the KNN algorithm, the red circle indicates a search range (that is, a neighborhood) that is centered on the red point and that includes *K* nearest neighbor historical samples. The size of the circle depends on the distance between the red point and the *K*th nearest historical sample. The similarity (based on the Euclidean distance) between the current feature vector (red point) and the feature vector (black point) of each historical sample in the circle is calculated, and weights are allocated based on the similarity. Finally, the query cardinality ($\hat{y}$) is predicted by weighted averaging.

Figure 7-30 KNN model example



- Cost estimation parameter adjustment

  a. I/O cost parameters (**seq_page_cost**/**random_page_cost**):

  Calculate the average I/O time based on the actual measurement (total I/O time/number of read pages) and the cost parameters based on the default ratio (**seq_page_cost**:**random_page_cost** = 1:4) and the following formulas: **seq_page_cost** = Average I/O time/2. **random_page_cost** = Average I/O time × 2.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

b. CPU cost parameters (**cpu_tuple_cost**, **cpu_index_tuple_cost**, and **cpu_operator_cost**):

- **cpu_tuple_cost** (main regression target): The linear regression model is used for calculation. This model is established for the SeqScan operator: Execution time = Number of processed tuples × (**cpu_tuple_cost** + Number of CPU operations required for each tuple's expression evaluation × **cpu_operator_cost**).
- **cpu_index_tuple_cost**: Considering that the performance of index scan is significantly affected by the sequence and the analysis is complex, this parameter is not directly regressed. Instead, it is set based on the regression result of **cpu_tuple_cost** and the default ratio using the formula **cpu_index_tuple_cost** = **cpu_tuple_cost**/2.
- **cpu_operator_cost**: It is not directly regressed. Considering that the provided number of operations per CPU may deviate greatly from the actual overhead and multi-variate regression is easily affected by data quality (such as sample homogenization), the calculation uses the default ratio and the formula is **cpu_operator_cost** = **cpu_tuple_cost**/4. This setting simplifies the SeqScan model to a one-variable regression without a constant term: Execution time = Number of tuples × **cpu_tuple_cost** × (1 + Number of CPU operations required for each tuple's expression evaluation/4).

c. Parameter reliability check:

- Training data is managed through sliding window management (the default value of **cost_update_window_size** is **5**).
- Each time the data in the window is completely updated, the goodness of fit of the new window data is calculated using the parameter trained last time. The coefficient of determination R^2 is calculated as follows: $R^2 = 1 - (\Sigma (\textit{estimated error}^2)/\textit{window length})$.
- The regression result is reliable and the new parameter is adopted only when $R^2 > 0.95$. Otherwise, data continues to be collected for subsequent training.

Performance Metrics

After this functionality is enabled, if the training of the cardinality estimation feedback model converges, the end-to-end query speed can be increased by 1.3 times on average in the Join-Order-Benchmark (JOIN) test set.

API Description

- Core system functions

Function	Description
gs_acm_analyze_workload_manual()	Manually start model training and update based on the collected operator information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

gs_stat_get_acm_feedback_operator_info()	View all operator information collected in the global memory.
gs_costmodel_calibration_manual()	Manually start collecting cost information and calibrate cost model parameters once.

For details about the description and related system functions, see "SQL Reference > Functions and Operators > AI Feature Functions" in *Reference*.

- Core GUC parameters

Parameter	Severity	Default Value	Description
enable_adaptive_cost	SIGHUP	on	Specifies whether to start or stop operator information collection, cardinality estimation feedback, and backend model maintenance thread.

enable_feedback_cardest	USRSET	on	Specifies whether to enable feedback estimation. If **enable_adaptive_cost** is disabled but **enable_feedback_cardest** is enabled, operator information is collected, and the cardinality estimation feedback API is still called. However, the backend automatic model maintenance thread is not started. You can manually train the model using the gs_acm_analyze_workload_manual() function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

adaptive_cardest_strategy	USRSET	auto	Sets the preference of the cardinality estimation model. The options are as follows: • **'auto'**: The model is selected based on the past prediction error. • **'use_statistics'**: Statistics are preferentially used for cardinality estimation. • **'use_feedback'**: The feedback model is preferentially used for cardinality estimation.
adaptive_cost_min_time	USRSET	1000ms	Specifies the SQL time threshold for operator collection. Only the execution operators of statements whose execution time is greater than the value of this parameter are collected.
cost_update_window_size	USRSET	5	Adjusts the size of the sliding window that collects data for adaptive cost model parameter regression. The value range is [1,20].

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

adaptive_costest_strategy	USRSET	L0	Sets the policy for using the new or old cost for cost estimation. Currently, there are two levels: • **'L0'**: If all cardinalities used for cost estimation of the plan tree are trusted (cardinalities of base table SeqScan/ cardinalities from feedback cardinalities/ cardinalities supporting feedback cardinalities but not trained/default cardinalities with an estimation error less than 1.1), the new cost is used for estimation and plan selection. • **'L1'**: The new cost model is used for estimation at any time.

For details about the description and related GUC parameters, see "Database GUC Parameters > GUC Parameters > AI Features" in *Reference*.

7.12.2.3 Operation Guide

Prerequisites

- The database is working properly.
- The feedback cardinality functionality is enabled.
- The database can be properly queried and parsed to start plan optimization.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Usage Guide

Enable the **enable_adaptive_cost** parameter and retain the default values of other parameters to enable this feature.

The cardinality estimation feedback feature requires that the cardinality be collected before it can correct statistical deviations. Cardinality information is collected only for SQL statements executed via EXPLAIN (ANALYZE) or with an execution time exceeding the value of **adaptive_cost_min_time**.

When **adaptive_cardest_strategy** is set to **auto** (default value), cardinality estimation feedback is not used for SQL statements of the same type (distinguished by SQL ID) during their initial execution. For SQL statements of the same type that are frequently executed, the system evaluates the feasibility of utilizing the estimated cardinality based on historical execution time. If a severe plan degradation occurs, namely, the current execution time surpasses the average execution time by tenfold without using cardinality estimation feedback and adds five more seconds, the corresponding SQL statement will be blacklisted and the cardinality estimation feedback will not be utilized. For instance, the average execution time of an SQL statement that does not utilize the cardinality estimation feedback is 1s. If the execution time surpasses 15s (1 x 10 + 5) after the cardinality estimation feedback is utilized, the SQL statement will be added to the blacklist.

Examples

The execution plan in the example may be different from the actual one due to differences in the environment, version, and deployment mode. This example is for reference only. The actual optimization effect is subject to the actual execution plan.

- Use the background thread to automatically train models and adjust execution plans.

Step 1 Start the database and check whether the **enable_adaptive_cost parameter** is enabled. Ensure that the background training of the cardinality estimation feedback functionality is enabled.

```
gaussdb=# SHOW enable_adaptive_cost;
 enable_adaptive_cost
----------------------
 on
(1 row)
```

Step 2 Set the plan display format.

```
gaussdb=# SET explain_perf_mode = pretty;
```

Step 3 Execute the statement that causes a suboptimal plan due to incorrect cardinality estimation, for example, a query SQL statement that contains multiple join paths and join types, and check whether a better plan is iterated after multiple executions of the execution plan. Use the JOB data set as an example. Import the JOB data set and run the query statement **01b.sql**.

```
gaussdb=# EXPLAIN (ANALYZE)
SELECT MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_title,
       MIN(t.production_year) AS movie_year
FROM company_type AS ct,
    info_type AS it,
    movie_companies AS mc,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    movie_info_idx AS mi_idx,
    title AS t
WHERE ct.kind = 'production companies'
  AND it.info = 'bottom 10 rank'
  AND mc.note NOT LIKE '%(as Metro-Goldwyn-Mayer Pictures)%'
  AND t.production_year BETWEEN 2003 AND 2010
  AND ct.id = mc.company_type_id
  AND t.id = mc.movie_id
  AND t.id = mi_idx.movie_id
  AND mc.movie_id = mi_idx.movie_id
  AND it.id = mi_idx.info_type_id;
```

Step 4 According to the execution plan, the total execution time is 1,442.612 ms. Pay attention to the **A-rows** (actual number of rows) and **E-rows** (estimated number of rows) columns. It can be found that the cardinality estimation during plan generation has significant deviations. For example, for the Hash Join(5, 11) operator, the actual number of rows is only 4, but the estimated number of rows is 528,329. Due to such cardinality estimation deviation, the execution plan is not the optimal choice. In this case, because the background training functionality has been enabled, the background training thread of the cardinality estimation feedback module is triggered to automatically execute the model training task.

```
 id |                                operation                                |       A-time        | A-rows  | E-rows  |
Peak Memory  | E-memory | A-width | E-width |  E-costs
----+-------------------------------------------------------------------------+---------------------+---------
+---------+---------------+----------+---------+---------+-----------
  1 | ->  Aggregate                                                           | 1442.612            |       1 |       1 |
29KB         |          |         |     113 | 107165.90
  2 |    ->  Streaming (type: GATHER)                                         | 1442.467            |       3 |       3 |
150KB        |          |         |     113 | 107165.90
  3 |       ->  Aggregate                                                     | [1382.897,1422.194] |       3 |       3 |
[27KB,28KB]  | 1MB      |         |     113 | 107165.78
  4 |          ->  Hash Join (5,11)                                           | [1382.845,1422.167] |       4 |
528329 | [10KB,10KB]  | 1MB      |         |      45 | 105844.94
  5 |             ->  Streaming(type: REDISTRIBUTE)                           | [435.768,481.418]   |  140904 |
1303008 | [69KB,69KB]  | 2MB      |         |      28 | 55550.75
  6 |                ->  Nested Loop (7,9)                                    | [495.739,559.512]   |  140904 |
1303008 | [4KB,4KB]    | 1MB      |         |      28 | 19718.03
  7 |                   ->  Streaming(type: BROADCAST)                        | [0.063,33.318]      |       3 |
3 | [45KB,45KB]  | 2MB      |         |       4 | 2.30
  8 |                      ->  Seq Scan on company_type ct                    | [0.003,0.074]       |       1 |
1 | [32KB,32KB]  | 1MB      |         |       4 | 2.02
  9 |                   ->  Bitmap Heap Scan on movie_companies mc             | [475.771,539.464]   |
140904 | 1303008 | [74KB,76KB]  | 1MB      |         |      32 | 15372.37
 10 |                      ->  Bitmap Index Scan using company_type_id_movie_companies |
[120.751,130.875]  | 1334883 | 1304564 | [356KB,356KB] | 1MB      |         |       0 | 3495.96
 11 |             ->  Hash                                                   | [833.688,891.113]   |       6 |  407937 |
[2MB,2MB]    | 16MB     | [48,52] |      29 | 44769.35
 12 |                ->  Hash Join (13,14)                                    | [833.646,891.058]   |       6 |
407937 | [11KB,11KB]  | 1MB      |         |      29 | 44769.35
 13 |                   ->  Seq Scan on title t                               | [704.619,725.190]   |  868629 |
872254 | [52KB,52KB]  | 1MB      |         |      25 | 24237.56
 14 |                   ->  Hash                                              | [13.300,84.206]     |      10 |  345009 |
[1MB,1MB]    | 16MB     | [20,20] |       4 | 15190.38
 15 |                      ->  Streaming(type: REDISTRIBUTE)                  | [13.221,84.122]     |      10 |
345009 | [45KB,45KB]  | 2MB      |         |       4 | 15190.38
 16 |                         ->  Nested Loop (17,19)                         | [10.662,46.745]     |      10 |
345009 | [3KB,3KB]    | 1MB      |         |       4 | 5702.64
 17 |                            ->  Streaming(type: BROADCAST)               | [10.177,46.329]     |       3
|       3 | [45KB,45KB]  | 2MB      |         |       4 | 1.75
 18 |                               ->  Seq Scan on info_type it              | [0.271,0.349]       |       1 |       1 |
[32KB,32KB]  | 1MB      |         |       4 | 1.48
 19 |                            ->  Bitmap Heap Scan on movie_info_idx mi_idx | [0.210,0.376]       |      10
|  345009 | [23KB,24KB]  | 1MB      |         |       8 | 4550.86
 20 |                               ->  Bitmap Index Scan using info_type_id_movie_info_idx | [0.160,0.271]       |
10 |  345009 | [3KB,3KB]    | 1MB      |         |       0 | 926.57
(20 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 5 Run the query again and observe the execution plan. The execution time has been optimized (from 6,763.152 ms to 5,337.537 ms). Analyze the cardinality estimation. Under the prediction of the cardinality estimation model, the cardinality estimation values of most operators are calibrated, optimizing the generation of the execution plan.

```
gaussdb=# EXPLAIN (ANALYZE)
SELECT MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_title,
       MIN(t.production_year) AS movie_year
FROM company_type AS ct,
     info_type AS it,
     movie_companies AS mc,
     movie_info_idx AS mi_idx,
     title AS t
WHERE ct.kind = 'production companies'
  AND it.info = 'bottom 10 rank'
  AND mc.note NOT LIKE '%(as Metro-Goldwyn-Mayer Pictures)%'
  AND t.production_year BETWEEN 2003 AND 2010
  AND ct.id = mc.company_type_id
  AND t.id = mc.movie_id
  AND t.id = mi_idx.movie_id
  AND mc.movie_id = mi_idx.movie_id
  AND it.id = mi_idx.info_type_id;
 id |                                      operation                                       |     A-time      | A-rows | E-rows | Peak
Memory | E-memory | A-width | E-width | E-costs
----+--------------------------------------------------------------------------------------+-----------------+--------
+--------+-------------+----------+---------+---------+---------
  1 | ->  Aggregate                                                                        | 72.986          |      1 |      1 |
29KB       |          |         |     113 | 5858.87
  2 |    ->  Streaming (type: GATHER)                                                      | 72.860          |      3 |      3 |
150KB      |          |         |     113 | 5858.87
  3 |       ->  Aggregate                                                                  | [55.949,56.995] |      3 |      3 |
[26KB,27KB] | 1MB      |         |     113 | 5858.74
  4 |          ->  Nested Loop (5,7)                                                       | [55.815,56.920] |      4 |  57132 |
[8KB,8KB]   | 1MB      |         |      45 | 5715.90
  5 |             ->  Streaming(type: BROADCAST)                                           | [0.099,0.197]   |      3 |      3
| [45KB,45KB] | 2MB      |         |       4 | 2.30
  6 |                ->  Seq Scan on company_type ct                                       | [0.001,0.038]   |      1 |      1 |
[32KB,32KB] | 1MB      |         |       4 | 2.02
  7 |             ->  Materialize                                                          | [55.498,56.637] |     81 |     19 |
[14KB,18KB] | 16MB     |         |      49 | 5713.55
  8 |                ->  Nested Loop (9,18)                                                | [55.427,56.565] |     78 |     19 |
[9KB,9KB]   | 1MB      |         |      49 | 5713.53
  9 |                   ->  Streaming(type: BROADCAST)                                     | [53.757,54.850] |     18 |
36 | [46KB,46KB] | 2MB      |         |      29 | 5705.90
 10 |                      ->  Nested Loop (11,17)                                         | [38.080,44.807] |      6 |     12 |
[6KB,6KB]   | 1MB      |         |      29 | 5704.81
 11 |                         ->  Streaming(type: REDISTRIBUTE)                            | [37.643,43.903] |     10 |
10 | [45KB,45KB] | 2MB      |         |       4 | 5702.88
 12 |                            ->  Nested Loop (13,15)                                   | [0.237,37.279]  |     10 |     10 |
[3KB,3KB]   | 1MB      |         |       4 | 5702.64
 13 |                               ->  Streaming(type: BROADCAST)                         | [0.074,37.080]  |      3 |
3 | [45KB,45KB] | 2MB      |         |       4 | 1.75
 14 |                                  ->  Seq Scan on info_type it                        | [0.080,0.153]   |      1 |      1 |
[32KB,32KB] | 1MB      |         |       4 | 1.48
 15 |                               ->  Bitmap Heap Scan on movie_info_idx mi_idx          | [0.113,0.138]   |     10 |
345009 | [23KB,24KB] | 1MB      |         |       8 | 4550.86
 16 |                                  ->  Bitmap Index Scan using info_type_id_movie_info_idx | [0.039,0.071]   |     10
| 345009 | [3KB,3KB]   | 1MB      |         |       0 | 926.57
 17 |                            ->  Index Scan using title_pkey on title t                | [0.310,0.761]   |      6 |      1 |
[26KB,26KB] | 1MB      |         |      25 | 0.63
 18 |                      ->  Index Scan using movie_id_movie_companies on movie_companies mc    |
[0.791,1.698]   |     78 |      2 | [16KB,16KB] | 1MB      |         |      32 | 0.62
(18 rows)
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Use the **EXPLAIN ADAPTCOST** option to display the model usage and use the execution plan in the previous part.

Step 1 Set the plan display format. The model usage can be displayed only in **normal** mode.

```
gaussdb=# SET explain_perf_mode = normal;
```

Step 2 Add the **ADAPTCOST** option to the EXPLAIN statement and execute the query.

```
gaussdb=# EXPLAIN (ANALYZE, ADAPTCOST)
SELECT MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_title,
       MIN(t.production_year) AS movie_year
FROM company_type AS ct,
     info_type AS it,
     movie_companies AS mc,
     movie_info_idx AS mi_idx,
     title AS t
WHERE ct.kind = 'production companies'
  AND it.info = 'bottom 10 rank'
  AND mc.note NOT LIKE '%(as Metro-Goldwyn-Mayer Pictures)%'
  AND t.production_year BETWEEN 2003 AND 2010
  AND ct.id = mc.company_type_id
  AND t.id = mc.movie_id
  AND t.id = mi_idx.movie_id
  AND mc.movie_id = mi_idx.movie_id
  AND it.id = mi_idx.info_type_id;
```

Step 3 Observe the execution plan output. The **cardest** field is added to the operator details. When the operator applies the cardinality estimation feedback model for cardinality prediction, this field displays the name of the loaded model. As shown in the third operator (Nested Loop), the value of the **cardest** field is **UMM_S_3988043888,17038.5**. This indicates the identifier name of the UMM that the operator uses for prediction. The model name is the feature value (hash value) of the operator.

```
                                                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------
--------------------------------
 Aggregate  (cost=5711.89..5711.96 rows=1 cardest=default width=113) (actual time=98.249..98.250 rows=1
loops=1)
   ->  Streaming (type: GATHER)  (cost=5711.89..5711.96 rows=3 cardest=default width=113) (actual
time=96.016..98.148 rows=3 loops=1)
         Node/s: All datanodes
         ->  Aggregate  (cost=5711.83..5711.84 rows=3 cardest=default width=113) (actual
time=[49.865,49.866]..[88.874,88.874], rows=3)
               ->  Nested Loop  (cost=955.32..5711.82 rows=4 cardest=trust UMM_S_3988043888,17038.5
width=45) (actual time=[49.777,49.812]..[88.842,88.842], rows=4)
                     Join Filter: (mc.company_type_id = ct.id), (Expression Flatten Optimized)
                     Rows Removed by Join Filter: 74
                     ->  Streaming(type: BROADCAST)  (cost=0.00..2.30 rows=3 cardest=default width=4) (actual
time=[0.070,0.075]..[0.099,0.105], rows=3)
                           Spawn on: All datanodes
                           ->  Seq Scan on company_type ct  (cost=0.00..2.02 rows=1 cardest=try default width=4)
(actual time=[0.002,0.002]..[0.050,0.052], rows=1)
                                 Filter: ((kind)::text = 'production companies'::text), (Expression Flatten Optimized)
                                 Rows Removed by Filter: 3
                     ->  Materialize  (cost=955.32..5709.24 rows=84 cardest=default width=49) (actual
time=[49.590,49.594]..[88.547,88.555], rows=81)
                           ->  Nested Loop  (cost=955.32..5709.17 rows=84 cardest=trust
UMM_S_229582469,17038.5(mixed) width=49) (actual time=[49.097,49.510]..[87.876,88.476], rows=78)
                                 Join Filter: (t.id = mc.movie_id), (Expression Flatten Optimized)
                                 ->  Streaming(type: BROADCAST)  (cost=955.32..5705.36 rows=18 cardest=default
width=29) (actual time=[48.944,48.971]..[87.581,87.619], rows=18)
                                       Spawn on: All datanodes
                                       ->  Nested Loop  (cost=955.32..5704.81 rows=6 cardest=trust
```

```
UMM_S_550930755,17038.5(mixed) width=29) (actual time=[0.227,30.510]..[77.161,77.197], rows=6)
                                  ->  Streaming(type: REDISTRIBUTE)  (cost=955.32..5702.88 rows=10
cardest=default width=4) (actual time=[0.073,30.357]..[76.971,76.982], rows=10)
                                        Spawn on: All datanodes
                                        ->  Nested Loop  (cost=955.32..5702.64 rows=10 cardest=trust
UMM_F_1223820966(mixed) width=4) (actual time=[0.188,0.985]..[34.022,73.538], row
s=10)
                                              ->  Streaming(type: BROADCAST)  (cost=0.00..1.75 rows=3
cardest=default width=4) (actual time=[0.074,0.867]..[33.869,73.378], rows=3)
                                                    Spawn on: All datanodes
                                                    ->  Seq Scan on info_type it  (cost=0.00..1.48 rows=1 cardest=try
default width=4) (actual time=[0.062,0.064]..[0.077,0.077], rows
=1)
                                                          Filter: ((info)::text = 'bottom 10 rank'::text), (Expression
Flatten Optimized)
                                                          Rows Removed by Filter: 112
                                              ->  Bitmap Heap Scan on movie_info_idx mi_idx
(cost=955.32..4550.86 rows=345009 cardest=default width=8) (actual time=[0.067,0.068]..[0
.093,0.097], rows=10)
                                                    Recheck Cond: (info_type_id = it.id), (Expression Flatten
Optimized)
                                                    ->  Bitmap Index Scan on info_type_id_movie_info_idx
(cost=0.00..926.57 rows=345009 cardest=default width=0) (actual time=[0.034,
0.034]..[0.047,0.047], rows=10)
                                                          Index Cond: (info_type_id = it.id)
                                        ->  Index Scan using title_pkey on title t  (cost=0.00..0.63 rows=1 cardest=try
default width=25) (actual time=[0.107,0.107]..[0.215,0.215], rows=6)
                                              Index Cond: (id = mi_idx.movie_id)
                                              Filter: ((production_year >= 2003) AND (production_year <= 2010))
                                              Rows Removed by Filter: 4
                                  ->  Index Scan using movie_id_movie_companies on movie_companies mc
(cost=0.00..0.62 rows=2 cardest=try default width=32) (actual time=[0.248,0.402]..[0.336,0.
632], rows=78)
                                        Index Cond: (movie_id = mi_idx.movie_id)
                                        Filter: (note !~~ '%(as Metro-Goldwyn-Mayer Pictures)%'::text)
                                        Rows Removed by Filter: 15
 Total runtime: 100.224 ms
(39 rows)
```

----End

7.12.2.4 Typical Issues

- This feature is used to generate plans. It is normal that the plans of the first several queries of the same type are different. After the convergence, a better plan is obtained.
- It is normal that this feature is not triggered when the cached plan is used. If no parameter is optimized, the system does not search for any feedback model.
- For SQL statements not executed using EXPLAIN (ANALYZE), if the plan generation time exceeds the execution time, the cardinality estimation feedback will not be utilized.

7.13 CBO

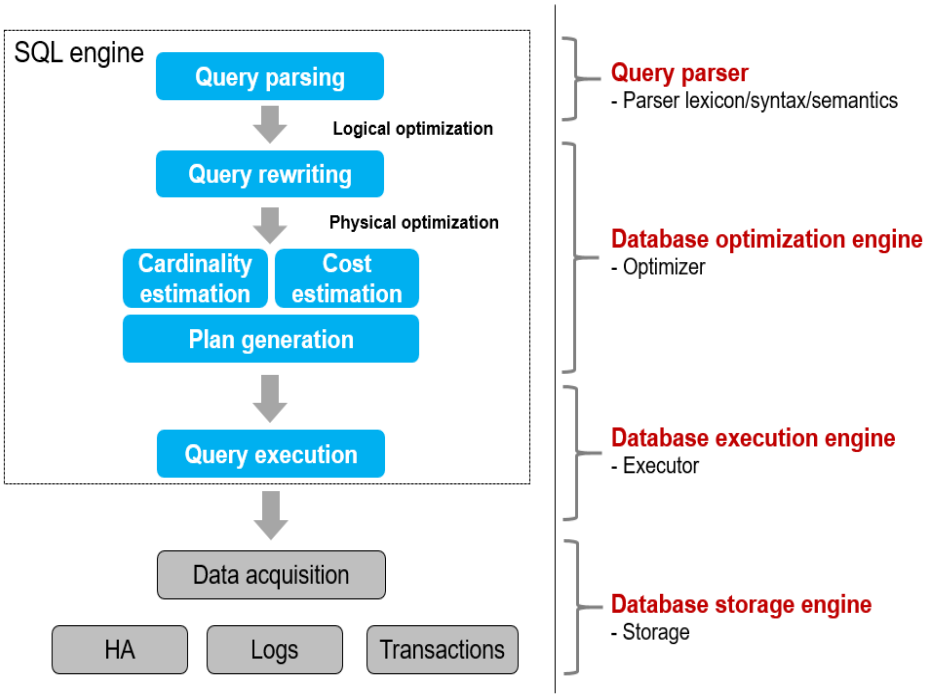
7.13.1 Overview

Optimizers are integral to the SQL engine. In the SQL engine, the original SQL statement is parsed by the parser to generate a query tree. Then, the optimizer performs logical optimization and physical optimization on the query tree to

generate a query plan that can be executed by the executor. Logical optimization mainly occurs in the rule model of the optimizer, and therefore is also referred to as rule-based optimizer (RBO). Its main functionality is to rewrite the query tree based on heuristic rules as an equivalent SQL statement that is possibly more efficient. Physical optimization occurs in the cost model of the optimizer, and therefore is also referred to as cost-based optimizer (CBO). Its main functionality is to generate different execution paths based on the query tree and select an execution mode based on the cost estimation, for example, whether the same table is accessed via full table scan or index scan, and whether different tables are joined via nested-loop join, hash join, or merge join.

An SQL statement can generate many paths that can be selected. CBO needs to calculate the cost of each optional path based on the statistics and cost model, and select the path with the minimum cost as the final execution path. **Figure 7-31** shows the position of CBO in the SQL engine.

Figure 7-31 SQL engine architecture



This feature focuses on the logical implementation principles of CBO in the SQL engine, mainly including statistics, scan path and connection path generation, cost estimation, and plan selection. The scan path and connection path determine the specific execution mode of SQL statements in the executor phase, and the statistics and cost estimation determine how to select a better execution mode from multiple generated paths.

7.13.2 Technical Principle

7.13.2.1 Statistics

The optimizer generates all possible access paths (execution plans) for SQL statements and selects the execution path with the lowest **COST** value. To enable

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the optimizer to accurately calculate the **COST** value of each execution plan, you need to obtain the description and characteristics of the data accessed during SQL statement execution, that is, optimizer statistics.

Optimizer statistics include the number of rows and the number of pages in a base table, number of distinct values (NDV), most common value (MCV), histogram, fraction of null values (nullfrac), and multi-column statistics.

7.13.2.1.1 Single-Column Statistics

Number of Rows and Number of Pages

The number of rows and the number of pages describe the statistics of the entire table. **reltuples** indicates the number of rows in the table, and **relpages** indicates the number of pages in the table. For an ordinary table, they can be queried from pg_class. For a partition of a partitioned table, they can be queried from pg_partition. The number of rows and the number of pages are used to estimate the size of query results. Generally, they are used together with other statistics (such as NDV, MCV, and histogram) to fully showcase the data distribution and storage status.

The following is an example of querying the number of rows and the number of pages:

```
-- The normal mode is used for displaying the plan of the "Statistics" section.
gaussdb=# SET explain_perf_mode = normal;
SET
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 1000), generate_series(1, 1000));
INSERT 0 1000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SELECT reltuples,relpages FROM pg_class where relname = 't1';
 reltuples | relpages
-----------+----------
      1000 |        5
(1 row)

gaussdb=# CREATE TABLE test_range_pt (a int, b int, c int)
PARTITION BY range(a)
(
PARTITION p1 VALUES LESS THAN(100),
PARTITION p2 VALUES LESS THAN (200),
PARTITION p3 VALUES LESS THAN (300),
PARTITION p4 VALUES LESS THAN (maxvalue)
)ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO test_range_pt VALUES(generate_series(1, 1000), generate_series(1, 1000));
INSERT 0 1000
gaussdb=# ANALYZE test_range_pt;
ANALYZE
gaussdb=# SELECT reltuples,relpages FROM pg_partition WHERE relname = 'p1';
 reltuples | relpages
-----------+----------
        86 |        1
(1 row)
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE test_range_pt;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

For system functions, the optimizer performs static cost estimation based only on pg_proc.prorows/procost and does not maintain statistics on ordinary tables. Therefore, the estimated number of rows may differ from the actual number of rows returned at runtime. When querying system functions, avoid using high-cost system function subqueries directly in complex joins, especially in combination with non-equality conditions such as LIKE, regular expressions, and function expressions. You are advised to first materialize system function results into temporary tables or CTEs before performing filtering and join operations. This prevents excessive query latency caused by the repeated execution of nested loops.

Number of Distinct Values (NDV)

NDV indicates the number of distinct values in a column of a database table. For example, in an employee table, if the gender column has only two options, male and female, the NDV of the column is 2.

For the database optimizer, accurate NDV information is critical to execution plan selection. During table join, if the NDV of the join condition column is low (that is, there are many duplicate values), the optimizer may prefer to select algorithms such as hash join. This is because for columns with a large number of duplicate values, hash join is more efficient when building hash tables.

NDV query example:

```
gaussdb=# CREATE TABLE Employee(name text, gender text);
CREATE TABLE
gaussdb=# INSERT INTO Employee VALUES('Ming','Male'),('Zhang','Male'),('Hua','Female'),('Li','Female');
INSERT 0 4
gaussdb=# ANALYZE Employee;
ANALYZE
gaussdb=# SELECT tablename,attname,n_distinct FROM pg_stats WHERE tablename='employee';
 tablename | attname | n_distinct
-----------+---------+------------
 employee  | name    |         -1
 employee  | gender  |        -.5
(2 rows)
gaussdb=# INSERT INTO Employee SELECT * FROM Employee;
INSERT 0 4
gaussdb=# INSERT INTO Employee SELECT * FROM Employee;
INSERT 0 8
gaussdb=# INSERT INTO Employee SELECT * FROM Employee;
INSERT 0 16
gaussdb=# ANALYZE Employee;
ANALYZE
gaussdb=# SELECT tablename,attname,n_distinct FROM pg_stats WHERE tablename='employee';
 tablename | attname | n_distinct
-----------+---------+------------
 employee  | name    |      -.125
 employee  | gender  |          2
(2 rows)
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# EXPLAIN SELECT * FROM Employee t1, Employee t2 WHERE t1.gender = t2.gender;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=8.41..32.22 rows=512 width=22)
   Node/s: All datanodes
   ->  Hash Join  (cost=4.41..8.22 rows=512 width=22)
         Hash Cond: (t1.gender = t2.gender)
         ->  Seq Scan on employee t1  (cost=0.00..1.18 rows=32 width=11)
         ->  Hash  (cost=4.01..4.01 rows=64 width=11)
               ->  Streaming(type: BROADCAST)  (cost=0.00..4.01 rows=64 width=11)
                     Spawn on: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                ->  Seq Scan on employee t2  (cost=0.00..1.18 rows=32 width=11)
(9 rows)

gaussdb=# DROP TABLE Employee;
DROP TABLE
```

The value of **n_distinct** can be:

- = 0: Unknown or not calculated.
- > 0: Number of distinct values after deduplication.
- < 0: The absolute value is the ratio of the number of distinct values after deduplication to the total number of values.

In the preceding example, the NDV of the **gender** column is 2, indicating that the column has only two values. During the query, the estimated number of joined rows is calculated as follows: 32 (total number of rows in the outer table) × 32 (total number of rows in the inner table)/2 (maximum NDV of the inner and outer tables) = 512.

Most Common Value (MCV)

MCV refers to a group of values that appear most frequently in a column of a database table. For example, in a sales record table, the product category column may have categories such as electronic products, clothing, and food. Electronic products and clothing appear most frequently. In this case, they are the MCVs of the column.

During query optimization, MCV information helps the optimizer better estimate the number of rows in the query result. If the columns involved in the query condition contain MCV information, the optimizer can calculate the approximate number of rows that meet the condition based on the frequency of the filter values in the MCV. In this way, the optimizer can better determine whether to perform an index scan or a full table scan. When data skew occurs, the optimizer determines whether to use skew optimization based on MCV.

The MCV must be at least 1.25 times the average data frequency.

MCV query example:

```
gaussdb=# CREATE TABLE Records (id int, Category text);
CREATE TABLE
gaussdb=# INSERT INTO Records VALUES(generate_series(1, 10), 'Electronic products');
INSERT 0 10
gaussdb=# INSERT INTO Records VALUES(generate_series(11, 20), 'Clothing');
INSERT 0 10
gaussdb=# INSERT INTO Records VALUES(21, 'Food'), (22, 'Others');
INSERT 0 2
gaussdb=# ANALYZE Records;
ANALYZE
gaussdb=# SELECT tablename,attname,most_common_vals,most_common_freqs FROM pg_stats WHERE
tablename='records';
 tablename | attname  | most_common_vals | most_common_freqs
-----------+----------+------------------+-------------------
 records   | id       |                  |
 records   | category | {Clothing, Electronic products} | {.454545,.454545}
(2 rows)

gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# EXPLAIN SELECT * FROM Records WHERE Category = 'Clothing';
                        QUERY PLAN
------------------------------------------------------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Streaming (type: GATHER)  (cost=0.31..1.61 rows=10 width=13)
   Node/s: All datanodes
   ->  Seq Scan on records  (cost=0.00..1.14 rows=10 width=13)
   Filter: (category = 'Clothing'::text)
(4 rows)

gaussdb=# DROP TABLE Records;
DROP TABLE
```

most_common_vals indicates the MCV, and **most_common_freqs** indicates the ratio of the MCV to the total data volume.

In the preceding example, the selectivity of **Clothing** in the MCV is 0.454545. During the query, the estimated number of rows = 22 (total number of rows) × 0.454545 (MCV selectivity) = 10.

Histogram

A histogram is a tool used to describe data distribution. In a database, it records the distribution of values in a column of a table, including the occurrence frequency of each value or data in a value range. For example, for the score column of a student, the histogram may divide the score into several ranges: 0–59, 60–69, 70–79, 80–89, and 90–100. Each range records the number of students whose scores fall within the range.

The histogram is an important basis for the database optimizer to estimate the number of rows in the query result. When the query conditions involve range query and fuzzy query, the optimizer uses the histogram to estimate the number of rows more accurately. For example, for the query **SELECT * FROM students WHERE score BETWEEN 90 AND 100;**, the optimizer checks the number of rows in the range of 90–100 in the histogram to estimate the size of the query result and select an appropriate execution plan.

Histograms can also help the optimizer avoid selecting inappropriate indexes or table join modes when processing columns with unevenly distributed data. For example, if most values of a column are in an interval and few values are in other intervals, the optimizer can select an access method that is more suitable for the data distribution based on the histogram information.

GaussDB uses a frequency histogram. The following is an example of histogram query:

```
gaussdb=# CREATE TABLE students (nameid int,  score int);
CREATE TABLE
gaussdb=# INSERT INTO students VALUES(generate_series(1, 10), generate_series(50, 54));
INSERT 0 10
gaussdb=# INSERT INTO students VALUES(generate_series(11, 50),generate_series(60, 79));
INSERT 0 40
gaussdb=# INSERT INTO students VALUES(generate_series(51, 90),generate_series(80, 89));
INSERT 0 40
gaussdb=# INSERT INTO students VALUES(generate_series(91, 100),generate_series(91, 100));
INSERT 0 10
gaussdb=# ANALYZE students;
ANALYZE
gaussdb=# SELECT tablename,attname, histogram_bounds FROM pg_stats WHERE tablename='students';
 tablename | attname
|
histogram_bounds

-----------+---------
+-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-----------------------------------------------------------------------------------------------------------------------------------
 students  | nameid  |
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,
77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100}
 students  | score   | {91,92,93,94,95,96,97,98,99,100}
(2 rows)

gaussdb=# \x
Expanded display is on.
gaussdb=#  SELECT * FROM pg_stats WHERE tablename='students';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
schemaname              | public
tablename               | students
attname                 | nameid
inherited               | f
null_frac               | 0
avg_width               | 4
n_distinct              | -1
n_dndistinct            | -1
most_common_vals        |
most_common_freqs       |
histogram_bounds        |
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,
77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100}
correlation             | .584266
most_common_elems       |
most_common_elem_freqs |
elem_count_histogram    |
partitionname           |
subpartitionname        |
-[ RECORD 2 ]----------
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
schemaname              | public
tablename               | students
attname                 | score
inherited               | f
null_frac               | 0
avg_width               | 4
n_distinct              | -.45
n_dndistinct            | -.690476
most_common_vals        |
{80,81,82,83,84,85,86,87,88,89,50,51,52,53,54,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79}
most_common_freqs       |
{.04,.04,.04,.04,.04,.04,.04,.04,.04,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,.02,
.02,.02,.02,.02}
histogram_bounds        | {91,92,93,94,95,96,97,98,99,100}
correlation             | .549571
most_common_elems       |
most_common_elem_freqs |
elem_count_histogram    |
partitionname           |
subpartitionname        |

gaussdb=# \x
Expanded display is off.
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# EXPLAIN SELECT * FROM students WHERE score > 90 AND score < 100;
                          QUERY PLAN
---------------------------------------------------------------
 Streaming (type: GATHER)   (cost=1.67..3.22 rows=10 width=8)
   Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  ->  Seq Scan on students  (cost=1.36..2.75 rows=10 width=8)
        Filter: ((score > 90) AND (score < 100))
(4 rows)

gaussdb=# EXPLAIN SELECT * FROM students WHERE score > 97 AND score < 99;
                        QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)  (cost=1.52..2.84 rows=2 width=8)
   Node/s: All datanodes
   ->  Seq Scan on students  (cost=1.46..2.75 rows=2 width=8)
         Filter: ((score > 97) AND (score < 99))
(4 rows)

gaussdb=# DROP TABLE students;
DROP TABLE
```

histogram_bounds stores the boundary values of each histogram bucket and generates the **values** array in ascending order. *values[0]* to *values[1]* determine the range of the first bucket, and so on. The frequencies of the intervals are approximately equal. This division method can more accurately reflect data distribution and effectively process extreme values and abnormal values.

The total selectivity of the histogram is not 1. Because there are MCVs in the preceding example, the sum of MCV selectivity is 0.9 and the total selectivity of the histogram is 1 – 0.9 = 0.1. When values within the range of 90–100 are queried, the estimated number of rows is calculated as follows: 100 (total number of rows) × 0.1 (total selectivity of the histogram) = 10. When values within the range of 97–99 are queried, since 97–99 occupies 2 intervals and there are 9 intervals in total, the estimated number of rows is calculated as follows: 100 (total number of rows) × 0.1 × 2/9 (ratio of the query to the histogram) = 2.

Fraction of Null Values (nullfrac)

nullfrac describes the null value ratio of the entire table. The following shows how to query:

```
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10));
INSERT 0 10
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 10));
INSERT 0 10
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SELECT tablename,attname, null_frac FROM pg_stats WHERE tablename='t1';
 tablename | attname | null_frac
-----------+---------+-----------
 t1        | a       |         0
 t1        | b       |        .5
(2 rows)

gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b is null;
                        QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.31..1.57 rows=10 width=8)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.00..1.10 rows=10 width=8)
         Filter: (b IS NULL)
(4 rows)
gaussdb=# DROP TABLE t1;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.13.2.1.2 Multi-Column Statistics

Concept

Multi-column statistics are used to calculate and obtain the data distribution information of multiple columns in a table as a whole. The statistics include the number of distinct values (NDV), most common values (MCV), fraction of null values (nullfrac), and most common null values (NULL_MCV).

Functionality

If an SQL statement references multiple column constraints, these columns are independent of each other. When single-column statistics are used for calculation, these constraints are split into single sub-constraints. However, in actual applications, it cannot be ensured that these constraints are independent of each other. As a result, the estimation error may be large. Multi-column statistics are derived to calculate the dependency between columns and their data distribution. For example, in a student information table of a physical education class, most students who take basketball are male, and most students who take dance are female. When a query is performed to search for male students who take basketball, single-column statistics may cause a large error. In this case, multi-column statistics are required.

Example:

```
gaussdb=# CREATE TABLE class (studentid int, course_name text, gender text);
CREATE TABLE
gaussdb=# INSERT INTO class VALUES(generate_series(1, 500),'Basketball','Male');
INSERT 0 500
gaussdb=# INSERT INTO class VALUES(generate_series(501, 1000),'Dance','Female');
INSERT 0 500
gaussdb=# ANALYZE class;
ANALYZE
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# EXPLAIN SELECT * FROM class WHERE course_name = 'Basketball';
                         QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..33.69 rows=500 width=15)
   Node/s: All datanodes
   ->  Seq Scan on class  (cost=0.00..10.25 rows=500 width=15)
   Filter: (course_name = 'Basketball'::text)
(4 rows)

gaussdb=# EXPLAIN SELECT * FROM class WHERE course_name = 'Basketball' AND gender = 'Male';
                              QUERY PLAN
----------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..23.22 rows=250 width=15)
   Node/s: All datanodes
   ->  Seq Scan on class  (cost=0.00..11.50 rows=250 width=15)
   Filter: ( (course_name = 'Basketball'::text) AND (gender = 'Male'::text) )
(4 rows)

-- View the statistics of a single column.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT * FROM pg_stats  WHERE tablename = 'class';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
--------------------
schemaname             | public
```

```
tablename               | class
attname                 | studentid
inherited               | f
null_frac               | 0
avg_width               | 4
n_distinct              | -1
n_dndistinct            | -1
most_common_vals        |
most_common_freqs       |
histogram_bounds        |
{1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250,260,270,280
,290,300,310,320,330,340,350,360,370,380,390,400,410,420,430,440,450,460,470,480,490,500,510,520,530,54
0,550,560,570,580,590,600,610,620,630,640,650,660,670,680,690,700,710,720,730,740,750,760,770,780,790,8
00,810,820,830,840,850,860,870,880,890,900,910,920,930,940,950,960,970,980,990,1000}
correlation             | .91529
most_common_elems       |
most_common_elem_freqs |
elem_count_histogram    |
partitionname           |
subpartitionname        |
-[ RECORD 2 ]----------
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
--------------------
schemaname              | public
tablename               | class
attname                 | course_name
inherited               | f
null_frac               | 0
avg_width               | 7
n_distinct              | 2
n_dndistinct            | 2
most_common_vals | {Basketball, Dance}
most_common_freqs       | {.5,.5}
histogram_bounds        |
correlation             | .989651
most_common_elems       |
most_common_elem_freqs |
elem_count_histogram    |
partitionname           |
subpartitionname        |
-[ RECORD 3 ]----------
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
--------------------
schemaname              | public
tablename               | class
attname                 | gender
inherited               | f
null_frac               | 0
avg_width               | 4
n_distinct              | 2
n_dndistinct            | 2
most_common_vals | {Female, Male}
most_common_freqs       | {.5,.5}
histogram_bounds        |
correlation             | -.415406
most_common_elems       |
most_common_elem_freqs |
elem_count_histogram    |
partitionname           |
subpartitionname        |

-- View the statistics of multiple columns.
gaussdb=# \x
Expanded display is off.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 'class';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals | most_common_freqs | most_common_vals_null | most_common_freqs_null | histogr
am_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------+------------------
+------------------+----------------------+-----------------------+--------
----------+---------------+------------------
(0 rows)
-- Collect statistics about multiple columns.
gaussdb=# ANALYZE class ((course_name,gender));
ANALYZE
gaussdb=# EXPLAIN SELECT * FROM class WHERE course_name = 'Basketball' AND gender = 'Male';
                              QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..34.94 rows=500 width=15)
   Node/s: All datanodes
   ->  Seq Scan on class  (cost=0.00..11.50 rows=500 width=15)
   Filter: ( (course_name = 'Basketball'::text) AND (gender = 'Male'::text) )
(4 rows)

-- View the statistics of multiple columns.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 'class';
-[ RECORD 1 ]----------+--------------------------
schemaname             | public
tablename              | class
attname                | 2 3
inherited              | f
null_frac              | 0
avg_width              | 11
n_distinct             | 2
n_dndistinct           | 2
most_common_vals       | {"Basketball,Dance"},{"Male,Female"}
most_common_freqs      | {.5,.5}
most_common_vals_null  |
most_common_freqs_null |
histogram_bounds       |
partitionname          |
subpartitionname       |

gaussdb=# DROP TABLE class;
DROP TABLE
```

In the preceding example, the inserted data shows that the gender can be determined based on the course name. If the multi-column statistics are not collected and the course name is basketball (**course_name = 'Basketball'**), the estimated number of rows is 1000 (total number of rows) × 0.5 (MCV ratio of basketball courses) = 500, which is consistent with the actual number of rows However, when both the course name and gender are queried (**course_name = 'Basketball' and gender = 'Male'**), the estimated number of rows is 1000 (total number of rows) × 0.5 (MCV ratio of basketball courses) × 0.5 (MCV ratio of males) = 250, which is inconsistent with the actual number of rows (500).

After the multi-column statistics are collected, the estimated number of rows is 1000 (total number of rows) × 0.5 (MCV ratio of basketball and male gender) = 500, which is consistent with the actual number of rows.

Calculation Method

The following example data describes the calculation process of statistics.

Example:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
CREATE TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# INSERT INTO t1 VALUES (1 , 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 2 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 2 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 2 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 3 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 3 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (1 , 3 , 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, 1 , 1 , 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, NULL, 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, NULL, 3 , 4);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (NULL, NULL, 3 , 4);
INSERT 0 1
gaussdb=# SELECT * FROM t1;
 a | b | c | d
---+---+---+---
 1 | 1 | 1 | 1
 1 | 1 | 1 | 1
 1 | 1 | 1 | 1
 1 | 1 | 1 | 1
 1 | 2 | 3 | 4
 1 | 2 | 3 | 4
 1 | 2 | 3 | 4
 1 | 3 | 3 | 4
 1 | 3 | 3 | 4
 1 | 3 | 3 | 4
   | 1 | 1 | 1
   | 1 | 1 | 1
   | 1 | 1 | 1
   | 1 | 1 | 1
   |   | 3 | 4
   |   | 3 | 4
   |   | 3 | 4
(17 rows)
gaussdb=# ANALYZE t1((a,b));
ANALYZE
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
-[ RECORD 1 ]----------+--------------------------
schemaname             | public
tablename              | t1
attname                | 1 2
inherited              | f
null_frac              | .176471
avg_width              | 8
n_distinct             | -.235294
n_dndistinct           | -.333333
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
most_common_vals       | {"{1,1,1}","{1,2,3}"}
most_common_freqs      | {.235294,.176471,.176471}
most_common_vals_null  | {"{NULL}","{1}"}
most_common_freqs_null | {.235294}
histogram_bounds       |
partitionname          |
subpartitionname       |

gaussdb=# DROP TABLE t1;
DROP TABLE
```

- **null_frac**: fraction of **NULL** values. There are three rows of [null,null], so the value of this parameter is calculated as follows: 3/17 (total number of rows) = 0.176471.
- **n_distinct**: number of distinct values. After **NULL** values are removed, there are four values [1,1], [1,2], [1,3], and [null,1], so the value of this parameter is calculated as follows: 4/17 (total number of rows) = 0.235294.
- **n_dndistinct**: average number of distinct values on each DN. The value cannot be accurately calculated because it is divided by the number of DNs, which causes some precision loss.
- **most_common_vals**: MCVs. None of the MCVs stored in this column is **NULL**. The values in the same position in each pair of parentheses ({}) falls into a group. In this example, the MCVs are [1,1], [1,2], and [1,3] in sequence.
- **most_common_freqs**: proportion of MCVs to the total number of rows. [1,1] has four rows: 4/17 = 0.23529. [1,2] has three rows: 3/17 = 0.176471. [1,3] has three rows: 3/17 = 0.176471.
- **most_common_vals_null**: MCVs that contain **NULL**. At least one of the MCVs stored in this column is **NULL**, but not all of them are **NULL**. Only [null,1] meets this condition.
- **most_common_freqs_null**: proportion of MCVs that contain **NULL** to the total number of rows. The MCVs with both **NULL** and non-**NULL** elements is only within the range of [null,1], which have four rows. Therefore, the proportion is 4/17 = 0.23529.
- **histogram_bounds**: histogram information. It does not exist in multi-column statistics.

Collection Methods

In GaussDB, by default, ANALYZE collects only statistics of a single column in a table and does not collect statistics of multiple related columns. You can view single-column statistics in the pg_stats view and multi-column statistics in the pg_ext_stats view. You can use any of the following methods to collect multi-column statistics:

- Manually specify column names.

  Example: ANALYZE t1((a,b));

```
gaussdb=# CREATE TABLE t1 (a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# ANALYZE t1;
ANALYZE
-- View the statistics of a single column.
gaussdb=# SELECT * FROM pg_stats  WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
most_common_vals    |        most_common_freqs         | histogram_bounds | correlation | mos
t_common_elems | most_common_elem_freqs | elem_count_histogram | partitionname |
subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-----------------------+-----------------------------+------------------+-------------+----
---------------+-----------------------+---------------------+---------------+------------------
 public     | t1        | a       | f         |         0 |         4 |        -.5 |          -.5 | {1,2,3,4,5,6,7,8,9,10} |
{.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |                |     .409023 |
            |                       |                     |               |
 public     | t1        | b       | f         |         0 |         4 |         2 |     -.166667 | {1,2}                  |
{.5,.5}                       |                |     .541353 |
            |                       |                     |               |
(2 rows)
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals | most_common_freqs | most_common_vals_null | most_common_freqs_null |
histogr
am_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+------------------+------------------+----------------------+-----------------------+--------
----------+---------------+------------------
(0 rows)

-- Collect statistics about multiple columns.
gaussdb=# ANALYZE t1((a,b));
ANALYZE
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct
|                  most_common_vals                  |        most_common_freqs         | most_com
mon_vals_null | most_common_freqs_null | histogram_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+----------------------------------------------------+----------------------------------+---------
--------------+------------------------+------------------+---------------+------------------
 public     | t1        | 1 2     | f         |         0 |         8 |        -.5 |          -.5 |
{"{1,2,3,4,5,6,7,8,9,10}","{1,2,1,2,1,2,1,2,1,2}"} | {.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |
            |                       |                |               |
(1 row)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

- Add multiple columns using the ALTER TABLE statement.

  For example, run **ALTER TABLE t1 ADD statistics ((a,b));** and then **ANALYZE t1;** to collect statistics of multiple columns. ALTER TABLE only reserves a slot for the multiple columns and does not collect statistics. To delete the statistics of multiple columns, run **ALTER TABLE t1 DELETE statistics((a,b))**.

```
gaussdb=# CREATE TABLE t1 (a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# ANALYZE t1;
ANALYZE
-- View the statistics of a single column.
gaussdb=# SELECT * FROM pg_stats  WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals    |        most_common_freqs         | histogram_bounds | correlation | mos
t_common_elems | most_common_elem_freqs | elem_count_histogram | partitionname |
subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-----------------------+-----------------------------+------------------+-------------+----
---------------+-----------------------+---------------------+---------------+------------------
 public     | t1        | a       | f         |         0 |         4 |        -.5 |          -.5 | {1,2,3,4,5,6,7,8,9,10} |
{.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |                |     .409023 |
            |                       |                     |               |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 public     | t1        | b       | f         |          0 |          4 |          2 |     -.166667 | {1,2}                 |
{.5,.5}                       |               |     .541353 |
            |                       |                   |             |
(2 rows)
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals | most_common_freqs | most_common_vals_null | most_common_freqs_null |
histogr
am_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+------------------+-------------------+-----------------------+------------------------+--------
----------+---------------+------------------
(0 rows)
-- Set multi-column statistics.
gaussdb=# ALTER TABLE t1 ADD statistics ((a,b));
ALTER TABLE
gaussdb=# ANALYZE t1;
ANALYZE
-- View the statistics of a single column.
gaussdb=# SELECT * FROM pg_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals    |        most_common_freqs         | histogram_bounds | correlation | mos
t_common_elems | most_common_elem_freqs | elem_count_histogram | partitionname |
subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-----------------------+-------------------------------+------------------+-------------+----
---------------+-----------------------+-------------------------+---------------+-----------------
 public     | t1        | a       | f         |          0 |          4 |         -.5 |          -.5 | {1,2,3,4,5,6,7,8,9,10} |
{.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |               |     .409023 |
            |                       |                   |             |
 public     | t1        | b       | f         |          0 |          4 |          2 |     -.166667 | {1,2}                 |
{.5,.5}                       |               |     .541353 |
            |                       |                   |             |
(2 rows)
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct
|                  most_common_vals                  |        most_common_freqs         | most_com
mon_vals_null | most_common_freqs_null | histogram_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+----------------------------------------------------+----------------------------------+---------
--------------+------------------------+------------------+---------------+------------------
 public     | t1        | 1 2     | f         |          0 |          8 |         -.5 |          -.5 |
{"{1,2,3,4,5,6,7,8,9,10}","{1,2,1,2,1,2,1,2,1,2}"} | {.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |
            |                       |                   |             |
(1 row)
-- Delete statistics of multiple columns.
gaussdb=# ALTER TABLE t1 DELETE statistics((a,b));
ALTER TABLE
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals | most_common_freqs | most_common_vals_null | most_common_freqs_null |
histogr
am_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+------------------+-------------------+-----------------------+------------------------+--------
----------+---------------+------------------
(0 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

- Create multi-column statistics automatically based on the index prefix.
  a. Set the **auto_statistic_ext_columns** parameter to a value within the range [1,4].

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

b. Create an index for a table. For example, create an index by running **CREATE INDEX idx1 ON t1(a,b,c,d);** for table **t1**.

c. Collect statistics.

```
gaussdb=# CREATE TABLE t1 (a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 2));
INSERT 0 10
gaussdb=# ANALYZE t1;
ANALYZE
-- View the statistics of a single column.
gaussdb=# SELECT * FROM pg_stats  WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals    |        most_common_freqs         | histogram_bounds | correlation | mos
t_common_elems | most_common_elem_freqs | elem_count_histogram | partitionname |
subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-----------------------+---------------------------------+------------------+-------------+----
---------------+-----------------------+----------------------+---------------+------------------
 public     | t1        | a       | f         |         0 |         4 |         -.5 |          -.5 | {1,2,3,4,5,6,7,8,9,10} |
{.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |                |     .409023 |
            |                   |                  |             |
 public     | t1        | b       | f         |         0 |         4 |          2 |     -.166667 | {1,2}                  |
{.5,.5}                      |                |     .541353 |
            |                   |                  |             |
(2 rows)
-- View the statistics of multiple columns.
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals | most_common_freqs | most_common_vals_null | most_common_freqs_null |
histogr
am_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+------------------+-------------------+-----------------------+------------------------+--------
----------+---------------+------------------
(0 rows)
-- View a parameter.
gaussdb=# SHOW auto_statistic_ext_columns;
 auto_statistic_ext_columns
----------------------------
 4
(1 row)
-- Set a parameter.
gaussdb=# SET auto_statistic_ext_columns = 2;
SET
-- Create a prefix index.
gaussdb=# CREATE INDEX idx1 ON t1(a,b);
CREATE INDEX
gaussdb=# ANALYZE t1;
ANALYZE
-- View the statistics of a single column.
gaussdb=# SELECT * FROM pg_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct |
most_common_vals    |        most_common_freqs         | histogram_bounds | correlation | mos
t_common_elems | most_common_elem_freqs | elem_count_histogram | partitionname |
subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-----------------------+---------------------------------+------------------+-------------+----
---------------+-----------------------+----------------------+---------------+------------------
 public     | t1        | a       | f         |         0 |         4 |         -.5 |          -.5 | {1,2,3,4,5,6,7,8,9,10} |
{.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |                |     .168421 |
            |                   |                  |             |
 public     | t1        | b       | f         |         0 |         4 |          2 |     -.166667 | {1,2}                  |
{.5,.5}                      |                |     .541353 |
            |                   |                  |             |
(2 rows)
-- View the statistics of multiple columns.
```

```
gaussdb=# SELECT * FROM pg_ext_stats WHERE tablename = 't1';
 schemaname | tablename | attname | inherited | null_frac | avg_width | n_distinct | n_dndistinct
|                most_common_vals                |        most_common_freqs         | most_com
mon_vals_null | most_common_freqs_null | histogram_bounds | partitionname | subpartitionname
------------+-----------+---------+-----------+-----------+-----------+------------+--------------
+-------------------------------------------------+----------------------------------+---------
--------------+------------------------+------------------+---------------+------------------
 public     | t1        | 1 2     | f         |         0 |         8 |       -.5 |          -.5 |
{"{1,2,3,4,5,6,7,8,9,10}","{1,2,1,2,1,2,1,2,1,2}"} | {.1,.1,.1,.1,.1,.1,.1,.1,.1,.1} |
              |                        |                  |               |
(1 row)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

7.13.2.2 Path Generation

As described in **Overview**, the CBO phase mainly completes path generation and plan selection. The main difference between the CBO phase and the RBO phase is that the optimization in the RBO phase is about performing equivalent transformation based on logical operations in SQL statements, while the CBO phase selects a "best" path tree from various generated paths based on the cost model. Each path in the path tree is finally mapped to a physical operator that can be executed by the executor. The executor executes these physical operators to implement the query logic. In GaussDB, these physical operators can be classified into two types: scan paths and join paths.

- Scan paths

  A scan path refers to the way a base table can be traversed, including the sequential scan path, index scan path, and bitmap scan path. Take a basic SQL statement as an example:

  ```
  SELECT * FROM t1;
  ```

  The generated execution plan is as follows:

  ```
                        QUERY PLAN
  ------------------------------------------------------------------
   Streaming (type: GATHER)  (cost=0.12..2.27 rows=5 width=16)
     Node/s: All datanodes
     ->  Seq Scan on t1  (cost=0.00..2.02 rows=5 width=16)
  (3 rows)
  ```

  In this example, "Seq Scan on t1" is generated by the sequential scan path. For different SQL statements, the optimizer can generate different scan paths based on different constraints and data distribution. For details, see **Scan Path Generation**.

- Join paths

  A join path mainly records the physical connection mode between various base tables, including nested-loop join path, hash join path, and merge join path. For details, see **Connection Path Generation**. The same SQL statement can implement the query logic despite different join modes. However, the execution efficiency varies according to the join path. Take a basic inner join scenario as an example. The SQL statement is as follows:

  ```
  SELECT * FROM t1 INNER JOIN t2 ON t1.b = t2.b;
  ```

  In this example, a hash join path can be generated. The plan is as follows:

  ```
                                  QUERY PLAN
  ------------------------------------------------------------------------------------
   Streaming (type: GATHER)  (cost=187.03..576.53 rows=5000 width=32)
     Node/s: All datanodes
     ->  Hash Join  (cost=183.03..368.15 rows=5000 width=32)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      Hash Cond: (t1.b = t2.b)
      ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
            Spawn on: All datanodes
            ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
      ->  Hash  (cost=162.20..162.20 rows=5001 width=16)
            ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                  Spawn on: All datanodes
                  ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)
```

A nested-loop join path can also be generated. The plan is as follows:

```
                              QUERY PLAN
-----------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..42220.27 rows=5000 width=32)
   Node/s: All datanodes
   ->  Nested Loop  (cost=0.00..42011.90 rows=5000 width=32)
         Join Filter: (t1.b = t2.b)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..170.53 rows=5000 width=16)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)
```

The cost of the plan generated based on the hash join path is much lower than that of the plan generated based on the nested-loop join path. An important task in the physical optimization phase is to generate different physical join paths and select the one with the lowest cost to improve the execution performance of the query plan.

In addition, many factors need to be considered when the optimizer actually selects a path. For example, the optimizer adjusts the cost benchmark value based on the disk type, CPU parameters and other factors, evaluates whether to select the index scan path or sequential scan path for scanning the base table considering factors such as the data distribution statistics, condition constraints, and selectivity, and determines the path startup cost and total cost. Therefore, multiple physical paths need to be reserved during path generation. During plan generation, the cost estimation model is used to make decisions.

7.13.2.2.1 Scan Path Generation

Scan paths include sequential scan path, index scan path, and bitmap scan paths. This section describes these scan paths.

- Sequential Scan

  Sequential scan is the most basic scan mode in GaussDB. This scan mode is to read the base table pages in sequence from disks, obtain all tuples in the table, filter the tuples based on the constraint conditions, and finally return the tuples that meet the filter condition based on the query statement. Therefore, the algorithmic complexity of this scan mode is O(N).

  The following uses an example to describe the principle of the sequential scan path. The statement for creating a base table is as follows:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1,2,3,4),(2,3,4,5),(3,4,5,6),(4,5,6,7),(5,6,7,8);
INSERT 0 5
gaussdb=# ANALYZE t1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ANALYZE
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# SET explain_perf_mode=normal;
SET
```

Run the following SQL statement:

```
gaussdb=# SELECT * FROM t1 WHERE t1.b = 3;
 a | b | c | d
---+---+---+---
 2 | 3 | 4 | 5
(1 row)
```

For the preceding SQL statement, the executor traverses table **t1** and returns the rows where the value of column **b** in table **t1** is **3**. The generated query plan is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t1 WHERE t1.b = 3;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..2.15 rows=1 width=16) (actual time=2.450..3.849 rows=1
loops=1)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.00..2.02 rows=1 width=16) (actual time=[0.006,0.006]..[0.179,0.179],
rows=1)
         Filter: (b = 3)
         Rows Removed by Filter: 4
 Total runtime: 4.367 ms
(6 rows)
```

In the preceding plan, **Seq Scan on t1** indicates that a sequential scan path is generated, and **Filter: (b = 3)** is the filter condition of the sequential scan. According to the actual execution of the preceding plan, the executor traverses the entire table **t1**, filters out four rows of data **Rows Removed by Filter: 4**, and returns one row of data **(actual time=2.450..3.849 rows=1 loops=1)**, which is the same as expected.

The statement for dropping the created table in the preceding example is as follows:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

- Index Scan

  In GaussDB, tables are physically stored as heap tables. When a data record is inserted, it may be stored on a new page on the physical disk or on a page that becomes empty after the tuple is deleted. This storage structure is simple, but it cannot use the ordered search algorithm to improve the execution efficiency in some scenarios with sorting conditions or equivalent filter conditions. Because data not is stored in order, the entire table needs to be scanned in sequence to find all tuples that meet the conditions.

  To improve the query performance of the database, GaussDB provides multiple index scan modes to meet the requirements for high performance in large data volume query scenarios. A common index organization mode is B/B + tree. It uses an ordered tree structure to reorganize data on disks and adds extra lookup directories, reducing the data search complexity to O(Log(N)) for data that contains filter conditions such as **b =?** in SQL statements.

  In GaussDB, to generate an index path, you need to add an index constraint on the base table column of the filter condition. Otherwise, the index path cannot be generated. Indexes are classified as primary key indexes, unique

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

indexes (with **NULL** values), and unique non-null indexes. The following uses a table creation statement as an example:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# SET explain_perf_mode=normal;
SET
```

Run the following SQL statement:

```
gaussdb=# SELECT * FROM t1 WHERE t1.b = 200;
 a  |  b  | c | d
-----+-----+---+---
 200 | 200 | 2 |
(1 row)
```

For the preceding SQL statement, the executor traverses table **t1** and returns the rows where the value of column **b** in table **t1** is **200**. The generated query plan is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t1 WHERE t1.b = 200;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
------
 Streaming (type: GATHER)  (cost=0.06..2.59 rows=1 width=16) (actual time=2.307..4.253 rows=1
loops=1)
   Node/s: All datanodes
   ->  Index Scan using idx_t1_b on t1  (cost=0.00..2.47 rows=1 width=16) (actual time=[0.033,0.033]..
[0.055,0.057], rows=1)
         Index Cond: (b = 200)
 Total runtime: 4.645 ms
(5 rows)
```

In the plan, **Index Scan using idx_t1_b on t1** indicates that an index scan path is generated, and the **idx_t1_b** index created on column **b** of table **t1** is used. **Index Cond: (b = 200)** indicates the index scan condition. According to the actual execution of the preceding plan, the executor does not traverse the entire table **t1**, but directly finds the rows where the value of column **b** is **200** through the index. Compared with the preceding sequential scan plan, the execution plan is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT /*+tablescan(t1)*/* FROM t1 WHERE t1.b = 200;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..28.96 rows=1 width=16) (actual time=3.046..3.272 rows=1
loops=1)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.00..28.84 rows=1 width=16) (actual time=[0.120,0.760]..[0.964,0.964],
rows=1)
         Filter: (b = 200)
         Rows Removed by Filter: 4999
 Total runtime: 3.593 ms
(6 rows)
```

The preceding query plan uses the HINT of tablescan() to forcibly specify the sequential scan mode. In this plan, the executor traverses the entire table **t1** and returns the rows where the value of column **b** is **200**. This scan mode

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

needs to filter out 4999 invalid rows. Therefore, its takes longer time than index scan.

Index scan uses the idea of trading space for time. Through extra index sorting of unordered data stored in the heap, the data is converted into an organized data structure on the index tree and establishes a mapping relationship with the data on the heap. In this way, the target data can be quickly found. In addition to ordinary index paths, GaussDB can generate some special types of index paths in some specific scenarios. For example, when the filter condition and projection of an SQL statement contain only index columns, all data required by the query can be obtained from the index. In this case, the TABLE ACCESS BY INDEX ROWID operation of the entire tuple is avoided, and an index-only scan path can be generated. The typical scenario is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT b FROM t1 WHERE t1.b = 200;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
----------
 Streaming (type: GATHER)  (cost=0.06..1.49 rows=1 width=4) (actual time=2.171..2.370 rows=1
loops=1)
   Node/s: All datanodes
   ->  Index Only Scan using idx_t1_b on t1  (cost=0.00..1.37 rows=1 width=4) (actual
time=[0.030,0.032]..[0.037,0.037], rows=1)
         Index Cond: (b = 200)
         Heap Fetches: 0
 Total runtime: 2.711 ms
(6 rows)
```

In the plan, **Index Only Scan using idx_t1_b on t1** indicates that the query path is generated by index-only scan. When searching for data, you only need to access the index node without obtaining data from the mapped tuple.

The statement for dropping the created table in the preceding example is as follows:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

- Bitmap Scan

  A bitmap scan is a special index scan. It constructs a bitmap set using index paths of multiple query conditions, and then combines the intersection and union operations of the bitmap to implement data merging. In this way, random heap table access that can only be scanned in sequence is changed to orderly heap table access using index conditions. The conditions for generating the index scan path are described above. On this basis, if SQL statements include multiple query restrictions that all allow index paths to be generated, these index paths can be used as the candidate paths for bitmap scan. If these restrictions are simple expressions connected by predicates such as AND and OR, the bitmap scan path can be generated. The following uses a table creation statement as an example:

  ```
  gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
  CREATE INDEX
  gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
  CREATE INDEX
  gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
  floor(random()*10));
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# SET explain_perf_mode=normal;
SET
gaussdb=# SET enable_bitmapscan=on;
SET
```

Run the following SQL statement:

```
gaussdb=# SELECT * FROM t1 WHERE t1.a = 200 OR t1.b = 300;
  a  |  b  | c | d
-----+-----+---+---
 200 | 200 | 5 |
 300 | 300 | 0 |
(2 rows)
```

For the preceding SQL statement, the executor traverses table **t1** and return the rows where the value of column **a** is **200** or the value of column **b** is **300**. The generated query plan is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t1 WHERE t1.a = 200 OR t1.b = 300;
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
---------------
 Streaming (type: GATHER)  (cost=2.78..3.96 rows=2 width=16) (actual time=2.289..2.498 rows=2
loops=1)
   Node/s: All datanodes
   ->  Bitmap Heap Scan on t1  (cost=2.72..3.83 rows=2 width=16) (actual time=[0.046,0.046]..
[0.097,0.098], rows=2)
         Recheck Cond: ((a = 200) OR (b = 300))
         ->  BitmapOr  (cost=2.72..2.72 rows=2 width=0) (actual time=[0.034,0.034]..[0.040,0.040],
rows=0)
               ->  Bitmap Index Scan on idx_t1_a  (cost=0.00..1.36 rows=1 width=0) (actual
time=[0.020,0.020]..[0.027,0.027], rows=1)
                     Index Cond: (a = 200)
               ->  Bitmap Index Scan on idx_t1_b  (cost=0.00..1.36 rows=1 width=0) (actual
time=[0.012,0.012]..[0.013,0.013], rows=1)
                     Index Cond: (b = 300)
 Total runtime: 2.890 ms
(10 rows)
```

As shown in the preceding example, the bitmap scan contains four different paths, including the bitmap heap, bitmap OR, bitmap AND, and index scan. In the plan, **Bitmap Heap Scan on t1** indicates that the plan is generated by the bitmap index scan path, and the **idx_t1_a** index created on column **a** and the **idx_t1_b** index created on column **b** of table **t1** are used. According to the actual execution of the preceding plan, the executor searches for index values that meet the Index Cond constraint on the two index paths, combines the index scan results using the bitmap OR operator, and obtains the result set that meets the **Recheck Cond: ((a = 200) OR (b = 300))** condition.

Compared with the sequential scan plan, the execution plan is as follows:

```
gaussdb=# EXPLAIN ANALYZE SELECT /*+tablescan(t1)*/* FROM t1 WHERE t1.a = 200 OR t1.b = 300;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..33.13 rows=2 width=16) (actual time=3.943..4.879 rows=2
loops=1)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.00..33.00 rows=2 width=16) (actual time=[0.164,1.066]..[1.265,1.265],
rows=2)
         Filter: ((a = 200) OR (b = 300))
         Rows Removed by Filter: 4998
 Total runtime: 5.165 ms
(6 rows)
```

The preceding query plan uses the HINT of tablescan() to forcibly specify the sequential scan mode. In this plan, the executor traverses the entire table **t1** and returns the rows where the value of column **a** is **200** or the value of column **b** is **300**. This scan mode needs to filter out 4998 invalid rows. Therefore, its takes longer time than bitmap OR scan.

The statement for dropping the created table in the preceding example is as follows:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

7.13.2.2.2 Connection Path Generation

The preceding sections have introduced several common base table scan paths, including sequential scan path, index scan path, and bitmap scan path. However, in actual SQL query scenarios, base table query is not the only operation involved. Instead, multiple tables often need to be joined in various modes to generate the final query result. Common join operations in SQL queries include INNER JOIN, LEFT JOIN, and SEMI JOIN. Therefore, GaussDB generates different join paths during the CBO phase. These paths refer to physical join paths (different from logical join operations such as INNER JOIN and LEFT JOIN), that is, specific algorithms that can be parsed by the executor. Common join path implementations are classified into three types: nested-loop join, hash join, and merge join. The generation process of join paths is a process of continuously trying the three types of paths.

When two base tables are joined in different data distribution, index conditions, and constraint scenarios, the performance of different physical paths varies in actual execution. None of them can maintain high performance in all scenarios. Therefore, you need to select a proper join path based on the cost to improve the execution efficiency.

To describe the generation modes of the three join paths, create the following database tables:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 10000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# CREATE TABLE t2(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_a ON t2(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t2_b ON t2(b);
CREATE INDEX
gaussdb=# INSERT INTO t2 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 10000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ANALYZE
gaussdb=# SET enable_fast_query_shipping = off;
SET
gaussdb=# SET explain_perf_mode=normal;
SET
```

- Nested-Loop Join Path

  The nested-loop join path is the most common and intuitive join mode among the three join modes. When generating a join path, the nested-loop join path selects a table as the driving table (or outer table) and the other table as the driven table (or inner table). Data is matched by performing double-loop traversal on the driving table and driven table. The time complexity is O(N*M). Take a nested-loop scenario of two tables as an example. Run the following SQL statement:

  ```
  gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.a != t2.a AND t1.b != t2.b;
                                  QUERY PLAN
  ------------------------------------------------------------------------------------------
   Streaming (type: GATHER)  (cost=4.00..4750131.35 rows=99980001 width=32)
     Node/s: All datanodes
     ->  Nested Loop Left Join  (cost=0.00..584297.98 rows=99980001 width=32)
           Join Filter: ((t1.a <> t2.a) AND (t1.b <> t2.b))
           ->  Seq Scan on t1  (cost=0.00..48.33 rows=10000 width=16)
           ->  Materialize  (cost=0.00..999.65 rows=30000 width=16)
                 ->  Streaming(type: BROADCAST)  (cost=0.00..949.65 rows=30000 width=16)
                       Spawn on: All datanodes
                       ->  Seq Scan on t2  (cost=0.00..48.33 rows=10000 width=16)
  (9 rows)
  ```

  The preceding example is a typical scenario of generating a nested-loop join plan. In this plan, double-loop traversal is performed with table **t1** as the driving table and table **t2** as the driven table. Each time a row of data is scanned from the outer table **t1**, the inner table **t2** is traversed once, and the join operation is performed on the rows that meet the **t1.a!= t2.a AND t1.b!= t2.b** matching condition. According to the generation principle of the nested-loop join path, this join mode can generate candidate paths in all types of join scenarios except the FULL JOIN scenario. Therefore, this join mode can be used as the last choice when other join paths cannot be generated. However, the algorithmic complexity indicates the performance of this join mode may not be the best in scenarios with a large amount of data, especially when the constraint filtering is good.

  There is another special form of the nested-loop join path, that is, the parameterized path. Assume that the condition for joining tables **t1** and **t2** is **t1.a = t2.a**. According to the predicate pushdown principle, the join condition cannot be pushed down to either the inner or outer table because the join condition **t1.a = t2.a** join both the outer and inner tables. The constraint condition can be applied only after tuples of both tables are obtained. In this case, even if an index exists in column **a** of table **t2**, no index path can be generated to accelerate data filtering. To achieve this, you can parameterize the condition column of the outer table to generate a special Param type. During the process of driving the inner table by the outer table, when the value of **t1.a** is set, it can be passed as a constant to the constraint condition **t1.a = t2.a**, which can be converted to **t2.a = Const**. In this case, the condition with a constant predicate can be pushed down to the inner table, an index path can be generated in table **t2**, and the index condition can be used for acceleration. The following SQL statements illustrate a typical scenario:

  ```
  gaussdb=# SET enable_hashjoin=off;
  SET
  gaussdb=# SET enable_mergejoin=off;
  SET
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..1417.36 rows=10000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..1000.74 rows=10000 width=32)
         ->  Seq Scan on t1  (cost=0.00..48.33 rows=10000 width=16)
         ->  Index Scan using idx_t2_a on t2  (cost=0.00..0.28 rows=1 width=16)
               Index Cond: (t1.a = a)
(6 rows)
```

To show the parameterized path, hash join and merge join are disabled. As shown in the preceding plan, the condition **t1.a = t2.a** is pushed down to the index condition of table **t2**. When each row of the outer table is executed, the value of column **a** in the row can be passed to the inner table as a constant. The inner table can apply the constant query condition to the index of table **t2** to quickly retrieve data.

- Hash Join Path

  The hash join path uses hash tables to accelerate join operations. The data types of the columns specified in the join condition can be hashed (for example, numeric types), and the join condition must be an equivalent condition. Take a hash join scenario of two tables as an example. Run the following SQL statement:

  ```
  gaussdb=# SET enable_hashjoin=on;
  SET
  gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a;
                              QUERY PLAN
  ------------------------------------------------------------------------------
   Streaming (type: GATHER)  (cost=93.99..600.78 rows=10000 width=32)
     Node/s: All datanodes
     ->  Hash Left Join  (cost=89.99..184.15 rows=10000 width=32)
           Hash Cond: (t1.a = t2.a)
           ->  Seq Scan on t1  (cost=0.00..48.33 rows=10000 width=16)
           ->  Hash  (cost=48.33..48.33 rows=9999 width=16)
                 ->  Seq Scan on t2  (cost=0.00..48.33 rows=10000 width=16)
  (7 rows)
  ```

  The preceding example is a typical scenario of generating a hash join path plan. In this scenario, values of columns **a** in tables **t1** and **t2** are integers, meeting the equivalent join constraint. Therefore, a hash join path can be generated. During actual execution, the outer table **t1** drives the inner table **t2** after a hash table is created for data in the inner table based on the specified column (column **a** in table **t2** in the example). For each data record in the outer table, the hash table is used to search for matching entries.

  According to the principle of the hash table, a hash join path is more efficient than a nested-loop join path. The nested-loop join plan is as follows:

  ```
  gaussdb=# SET enable_hashjoin=off;
  SET
  gaussdb=# SET enable_mergejoin=off;
  SET
  gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a;
                                QUERY PLAN
  -------------------------------------------------------------------------------
   Streaming (type: GATHER)  (cost=4.00..1417.36 rows=10000 width=32)
     Node/s: All datanodes
     ->  Nested Loop Left Join  (cost=0.00..1000.74 rows=10000 width=32)
           ->  Seq Scan on t1  (cost=0.00..48.33 rows=10000 width=16)
           ->  Index Scan using idx_t2_a on t2  (cost=0.00..0.28 rows=1 width=16)
                 Index Cond: (t1.a = a)
  (6 rows)
  ```

  To ensure the generation of a nested-loop join path plan, the logic for generating the hash join and merge join paths is disabled in this example. The

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

comparison between the two plans shows that the plan generated by the hash join path incurs obviously higher cost than that generated by the hash join path.

- Merge Join Path

  The merge join path uses pre-sorting to accelerate join operations. According to the join condition, the base table is sorted by the column specified in the join condition. Then, the entries (usually equivalent conditions) that meet the join condition are found and output in the same way as the merge sort. Take a merge join scenario of two tables as an example. Run the following SQL statement:

```
gaussdb=# SET enable_mergejoin=on;
SET
gaussdb=# SET enable_hashjoin=off;
SET
gaussdb=# EXPLAIN SELECT t1.a,t2.a FROM t1 LEFT JOIN t2 ON t1.a = t2.a;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..589.11 rows=10000 width=8)
   Node/s: All datanodes
   ->  Merge Left Join  (cost=0.00..172.49 rows=10000 width=8)
         Merge Cond: (t1.a = t2.a)
         ->  Index Only Scan using idx_t1_a on t1  (cost=0.00..61.24 rows=10000 width=4)
         ->  Index Only Scan using idx_t2_a on t2  (cost=0.00..61.24 rows=10000 width=4)
(6 rows)
```

  Generally, the performance of a merge join path is between those of the hash join and nested-loop join paths. However, in some scenarios, the merge join path can bring obvious advantages. According to the preceding plan, the merge join path mainly uses the merge sort principle. This is because the SQL statement outputs only columns **a** of tables **t1** and **t2**, the columns are distribution keys, and both tables have indexes on the columns. The order of indexes can improve the performance of the merge sort algorithm, which can bring higher execution efficiency than hash join.

The statement for dropping the created tables in the preceding example is as follows:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

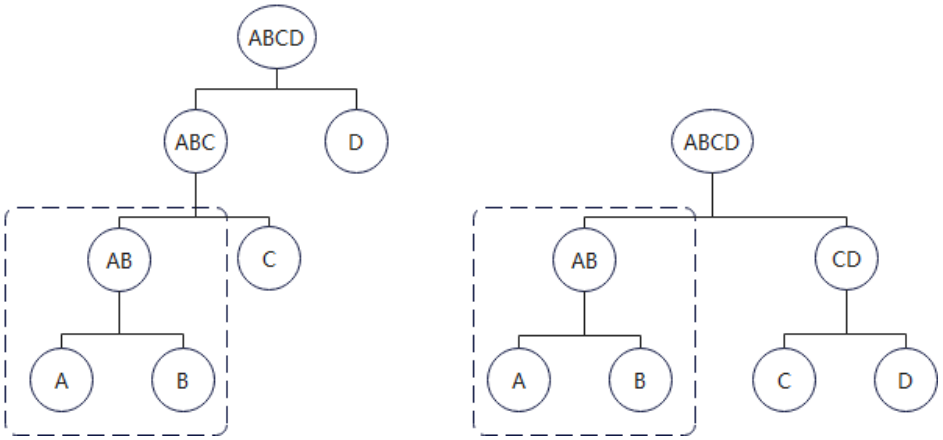
The preceding describes three common join paths in GaussDB. In actual scenarios, you need to consider the join order between tables in addition to join algorithms. Take INNER JOIN as an example. For a join relationship such as **t1 INNER JOIN t2**, when a nested-loop join path is generated, either table **t1** or **t2** can be used as the driving table. Nested-loop join is suitable for the scenario where a small table drives a large table. In this case, you need to select a driving table based on the data distribution of tables **t1** and **t2**. Therefore, generating a join path requires the comparison of different join order combinations between tables. Due to the complexity of SQL, the path generation process usually cannot be implemented through exhaustive method. For example, in the simplest two-table join scenario, at least 18 different paths need to be tried (3 join algorithms × 2 join orders × 3 scan paths). The actual situation is more complex. You also need to consider whether parameterized paths exist and whether constraints can be pushed down. Therefore, for multi-table join, the path scale increases exponentially. Therefore, intelligent algorithms are usually used in databases to prune paths and avoid

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

search space explosion. The common path generation algorithms are dynamic programming and genetic algorithms. Take the most widely used dynamic programming algorithm as an example. The core idea is to divide large problems into small problems, solve the small problems, store the optimal solutions of the small problems, and use these optimal solutions to construct the optimal solutions of large problems to avoid repeated calculation. The core ideas of the algorithm are as follows:

- Extract repeated subproblems.

  Figure 7-32 Extracting subproblems

  As shown in the preceding figure, the AXB join operation in the two join trees is a repeated subproblem. By continuously obtaining the lowest-cost path of each subproblem for a join tree and pruning the higher-cost ones, the process repeats iteratively layer by layer until the lowest-cost path of the complete join tree is revealed.

- Obtain the optimal substructure.

  For path generation, each leaf node is a table scan path. Even if the current scan path is the optimal, it does not necessarily mean that the subsequent join path is optimal. For example, the cost of sequential scan is less than the cost of index scan, and the cost of sequential scan plus the cost of sorting is greater than that of index scan. Therefore, when calculating the cost of a subproblem, the optimizer needs to save the optimal and suboptimal solutions, one of which may be the optimal for the startup cost path, parameterized path, or other paths. Dynamic programming can be implemented in two methods: recursion and iteration. Currently, the database uses the iteration method. It attempts to find the optimal subpath for two tables, and then iterates the process for three tables, four tables, and so on.

7.13.2.3 Plan Generation

The preceding sections describe the generation of various scan paths and join paths. The CBO filters these paths based on the cost and retains the optimal path. The filtering process consists of two steps. First, the optimizer determines whether to create a join path based on the initial cost. Many join paths can be generated based on different algorithms, join orders, and underlying scan methods. Therefore, the purpose of preliminary filtering is to filter out poorer paths to save the time for generating an execution plan. However, because the types of scan

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

paths are limited (there are only three basic types: sequential scan, index scan, and bitmap scan), filtering is usually not performed, leaving more options for join paths. Second, new costs are added to the path nodes after the preliminary filtering, such as the cost of expression processing and the cost of the LIMIT constraint. In this way, a final plan for comparing paths is generated. Then, the optimal path is selected based on factors such as the startup cost and total cost. After the optimal path is selected, it is further converted to a plan tree suitable for the executor to execute a complete plan.

7.13.2.3.1 Cost Estimation

To select the optimal execution plan, the optimizer needs criteria to measure execution plans. The execution time is an intuitive criterion for comparing different query plans. Generally, a query plan with a shorter execution time is better. However, the execution time of the same query plan varies according to environment variables. For example, the machine configuration (CPU frequency, memory size, and disk media) affects the actual execution performance. Therefore, a dimensionless value is required to replace the time. In GaussDB, the cost is an estimated value, serving as an absolute indicator to evaluate the advantages and disadvantages of different execution plans. The cost of each operator is calculated based on a specific formula, which is called a cost model. The cost of an operator can be calculated by substituting statistics into the cost model. This can be abstracted into the following formula:

$$Cost = F_{operator}(X_{statistics})$$

As shown in the preceding formula, the cost model of each operator is the abstraction of its execution logic. For each operation (such as page access and expression operation) that can be used as a basic calculation unit, the operation must be rendered dimensionless and represented in the unit of cost. Take sequential scan as an example. The execution logic is as follows: Pages are read from the file in sequence. If no predicate is available, the cost of sequential scan is the cost of scanning all pages. In this case, the cost of accessing a single page needs to be defined as the unit for calculating multiple pages in sequential scan.

To define the evaluation benchmarks of different factors, some key factors that affect the execution efficiency are extracted and abstracted into standard values. The costs of all paths are calculated based on the standard values. For example, the CPU frequency parameter, disk page I/O mode, and memory I/O mode are standardized as the benchmarks for cost estimation. The following describes the disk I/O and CPU benchmark costs.

- Benchmark cost of disk page I/O

  Disk page I/O modes include sequential read/write and random read/write. Take a traditional hard disk drive (HDD) as an example. An HDD needs to seek tracks for reading data, which takes some time. Therefore, a large amount of time is wasted on seeking during random read/write, whereas sequential read/write can save the seek time and improve I/O performance. In addition, disks usually have caches. During sequential read/write, disks can prefetch data to improve the cache hit ratio, so this mode better enhances performance compared with random read/write.

  Based on the different read/write modes, different cost benchmarks needs to be defined for sequential read/write and random read/write. In earlier

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

GaussDB versions, the default cost benchmark value of sequential read/write is **1.0**, and that of random read/write is **4.0**. However, with the popularization of solid-state drives (SSDs), the performance of random read/write of SSDs has been greatly improved, and the benchmark value is no longer accurate. Therefore, in the latest version, the default value of random read/write has been changed to **1.1**. The two benchmark values are not fixed in different production environments. Therefore, GaussDB allows you to modify GUC parameters based on the environment. The GUC parameter for adjusting the benchmark value of page sequential read/write is **seq_page_cost**, and the one for adjusting the benchmark value of page random read/write is **random_page_cost**.

In addition, the database has a cache system. If data is already in the cache, the disk I/O overhead can be avoided. Therefore, during cost estimation, simply calculating the disk I/O cost for each page is not enough. The number of cached pages also needs to be estimated during cost estimation. The number is determined by the GUC parameter **effective_cache_size**.

- CPU benchmark cost of tuples and restrictions

  After data is read from disk pages, it needs to be presented in the form of tuples. Therefore, CPU compute resources are required, and the CPU computational cost is generated. The compute resources consumed by tuples vary with the tuple organization structure and expression calculation complexity. Therefore, in GaussDB, the CPU computational costs of ordinary tuples, index tuples, and different data organization forms (such as projections and expressions) are distinguished, and the benchmark costs can be adjusted using GUC parameters.

  GUC parameter for controlling the CPU benchmark of ordinary tuples: **cpu_tuple_cost**. The default value is **0.01**.
  GUC parameter for controlling the CPU benchmark of index tuples: **cpu_index_tuple_cost**. The default value is **0.005**.
  GUC parameter for controlling the CPU benchmark of expressions: **cpu_operator_cost**. The default value is **0.0025**.

  In addition to the preceding benchmark costs, the startup cost and total cost need to be considered in GaussDB when comparing costs. The startup cost refers to the cost from the start of SQL statement execution to the time when the executor queries and obtains the first tuple. The total cost refers to all costs from the start to the end of SQL statement execution. Therefore, the following formula is used:

  total_cost = **startup_cost** + **run_cost**

  The startup cost varies according to the scan path and join path. Therefore, the startup cost needs to be estimated separately. The following uses a table creation statement as an example:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SET explain_perf_mode=normal;
SET
```

  Executing the same SQL statement generates different path plans.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE t1.a >= 100 ORDER BY a;
                                QUERY PLAN
---------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..252.12 rows=4900 width=16)
   Merge Sort Key: a
   Node/s: All datanodes
   ->  Index Scan using idx_t1_a on t1  (cost=0.00..47.99 rows=4900 width=16)
         Index Cond: (a >= 100)
(5 rows)
gaussdb=# EXPLAIN SELECT /*+tablescan(t1)*/* FROM t1 WHERE t1.a >= 100 ORDER BY a;
                            QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=119.99..324.19 rows=4899 width=16)
   Merge Sort Key: a
   Node/s: All datanodes
   ->  Sort  (cost=115.99..120.07 rows=4899 width=16)
         Sort Key: a
         ->  Seq Scan on t1  (cost=0.57..28.84 rows=4900 width=16)
               Filter: (a >= 100)
(7 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

In the preceding example, the startup cost of the same SQL statement varies with the path used in a plan. The index path is selected for the first execution. The statement needs to be sorted, so index path is implemented using a B+ tree to quickly locate the target data and ensure the order. The startup cost is 0. The sequential scan path is selected for the second execution. The first valid data record needs to be found by traversing the disk. The startup cost is 0.57. The total cost is higher than that of the index scan path due to the additional sorting operation. In this example, by default, the database eliminates the plan using both sequential scan and sorting based on the plan selection policy. To facilitate comparison, the hint of table scan is used to forcibly select the sequential scan plan.

Based on the preceding example, the cost calculation formula of the sequential scan path is as follows:

startup_cost = (**cpu_tuple_cost** + **cpu_expr_cost**) × **M<tuple>** + **seq_page_cost** × **M<page>**
run_cost = **cpu_run_cost** + **disk_run_cost** = (**cpu_tuple_cost** + **cpu_expr_cost**) × **N<tuple>** + **seq_page_cost** × **N<page>**

The values of **seq_page_cost** and **cpu_tuple_cost** come from the parameter settings of the preceding benchmark cost. In this example, the values are **1.0** and **0.01**, respectively. **N<tuple>** and **N<page>** indicate the total number of tuples and pages in the table, respectively. You can run the following SQL statement to obtain the values:

```
gaussdb=# SELECT reltuples, relpages FROM pg_class WHERE relname='t1';
 reltuples | relpages
-----------+----------
      5000 |       24
(1 row)
```

The value of **cpu_expr_cost** is the sum of the calculation costs of all expressions in the WHERE condition. In this example, the expression contains only the **t1.a >= 100** condition. The column **a** in table **t1** is of the INT type, and the expression calculation function is GE (greater than or equal to). Therefore, you can run the following SQL statement to obtain the calculation benchmark cost coefficient:

```
gaussdb=# SELECT proname, procost FROM pg_proc WHERE proname = 'int4ge';
 proname | procost
---------+---------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 int4ge |      1
(1 row)
```

The benchmark cost of the expression type is controlled by the GUC parameter **cpu_operator_cost**. In this example, the default value is **0.0025**. The preceding values are substituted into a formula, yielding the result as follows:

```
run_cost = (cpu_tuple_cost + cpu_expr_cost) × N<tuple> + seq_page_cost × N<page>
                              = (0.01+1x0.0025)*5000+1.0*24
                              = 86.5
```

However, in the preceding example, the actual cost is **28.84**. This is because the database in the test case scenario has three DNs. You can run the following SQL statement to query the information about the CN and DNs:

```
gaussdb=# SELECT node_name, node_type, hostis_primary FROM PGXC_NODE;
  node_name   | node_type | hostis_primary
--------------+-----------+----------------
 coordinator1 | C         | t
 datanode1    | D         | t
 datanode2    | D         | t
 datanode3    | D         | t
(4 rows)
```

As shown in the preceding query result, the data in table **t1** is evenly distributed on three DNs. Therefore, the cost of each DN is the average value, that is, 86.5/3 = 28.84.

The startup cost is calculated as follows. The sequential scan path is selected, that is, the entire data table is traversed. According to the definition of the startup cost and the assumption that data is evenly distributed, the 5000 data records are evenly distributed on 24 pages. The cost of obtaining the first useful data record is as follows:

```
startup_cost = (cpu_tuple_cost + cpu_expr_cost) × M<tuple> + seq_page_cost × M<page>
                                 = (0.01+1*0.0025) *100+1.0*(24/5000*100)
                                 = 1.25+0.48
                                 = 1.73
```

Similarly, because the data is evenly distributed on three DNs, the startup cost of a node is calculated as follows: 1.73/3 = 0.57.

The sequential scan scans the entire table, and the **run_cost** calculation includes the startup cost. Therefore, **total_cost** has the same value as **run_cost**.

7.13.2.3.2 Plan Selection

The preceding sections describe several common path generation modes in the database and the cost estimation model principles of the optimizer. Based on the preceding principles, the CBO can select the final execution plan. In the path generation phase, the optimizer uses dynamic programming and genetic algorithms to try different join paths and marks cost information for these paths. The optimizer only needs to compare the paths based on the cost and select the path with a lower cost. In this way, an optimal path tree is obtained.

During path selection, pay attention to the following factors:

- Startup cost

  When the LIMIT constraint is included, the path startup cost is a very important factor because the LIMIT constraint can terminate data traversal in advance.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Total cost

  The total cost of a path is the most important criterion.

- Path filtering performance

  If a path can filter most data in advance, the path delivers better filtering performance and returns fewer rows to the upper layer, reducing the computing workload.

- Whether there are hints

  Using hints is an important optimization method in the database and can serve as a compensation measure when the cost estimation is inaccurate. Therefore, paths with specific hints are more likely to be selected. This scenario is not discussed in this section.

Based on the preceding factors, the general principles for path selection are as follows (path A is selected and path B is new):

1. If the total cost of path A is lower than that of path B, check whether the startup cost of path B is lower than that of path A. If the startup cost of path B is lower than that of path A, retain path A and accept path B as a candidate path. Path B is retained because if some constraints such as LIMIT are used when the upper-layer plan is generated, path B may deliver better performance. During the entire path selection process, the optimizer does not retain only one optional path. If the startup cost of path B is not lower than that of path A, path B is discarded.
2. If the total cost of path B is lower than that of path A, but the startup cost of path A is lower than that of path B, retain path A and accept path B as a candidate path. If the startup cost of path A is not lower than that of path B, accept path B as the new candidate path and discard path A.
3. If the total cost difference between path A and path B is small, compare their startup costs and retain the path with a lower startup cost. If the startup cost difference is not large, check which path has better filtering performance. The path with fewer estimated returned rows is retained.

The preceding principles are the general principles of path selection. In actual applications, the comparison logic is more complex. For example, in a scenario where a parallel plan can be generated, factors such as data distribution need to be considered. In a scenario where a path parameter exists, the number of path parameters needs to be further compared. However, the overall policy is based on cost information. The CBO continuously eliminates poor paths through multi-layer iteration and finally selects a complete optimal path tree. However, this path tree contains a large amount of information that is not required by the executor and lacks processing of constraints such as AGG functions and LIMIT. Therefore, the optimizer supplements some operator logic and converts it into a plan tree that can be understood by the executor, that is, the final plan tree. In GaussDB, you can run the **EXPLAIN** command to print the plan tree. The following is an example of a simple sequential scan plan:

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE t1.a > 10;
                          QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.05..236.84 rows=4991 width=16)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.05..28.84 rows=4991 width=16)
         Filter: (a > 10)
(4 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

This example shows a complete plan, including the final plan scan path (**Seq Scan on t1**), filter condition (**Filter: (a > 10)**), and cost information (**cost=0.05..28.84 rows=4991 width=16**), such as the startup cost, total cost, and estimated number of returned rows.

The following describes common plan types and operators in GaussDB. The statements for creating a table are as follows:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# CREATE TABLE t2(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_a ON t2(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t2_b ON t2(b);
CREATE INDEX
gaussdb=# INSERT INTO t2 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# CREATE TABLE t3(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t3_a ON t3(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t3_b ON t3(b);
CREATE INDEX
gaussdb=# INSERT INTO t3 (a, b, c) VALUES (generate_series(1, 100), generate_series(1, 50),
floor(random()*10));
INSERT 0 100
gaussdb=# ANALYZE t3;
ANALYZE
gaussdb=# SET explain_perf_mode=normal;
SET
gaussdb=# SET enable_nestloop=off;
SET
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=off;
SET
```

- Distributed Stream plan:

```
gaussdb=# EXPLAIN SELECT count(t1.a),t1.c FROM t1 LEFT JOIN t2 ON t1.b = t2.b WHERE t1.a >
ALL(SELECT a FROM t3) GROUP BY t1.c ORDER BY t1.c LIMIT 10;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
-
 Limit  (cost=5260.77..5260.78 rows=5 width=16)
   ->  Sort  (cost=5260.77..5260.78 rows=5 width=16)
         Sort Key: t1.c
         ->  HashAggregate  (cost=5260.35..5260.71 rows=5 width=16)
               Group By Key: t1.c
               ->  Streaming (type: GATHER)  (cost=5260.35..5260.71 rows=15 width=16)
                     Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
               ->  HashAggregate  (cost=5260.04..5260.09 rows=15 width=16)
                     Group By Key: t1.c
                     ->  Hash Left Join  (cost=364.96..5243.37 rows=10000 width=8)
                           Hash Cond: (t1.b = t2.b)
                           ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..4836.74 rows=5000 width=12)
                                 Spawn on: All datanodes
                                 ->  Index Scan using idx_t1_b on t1  (cost=0.00..4699.21 rows=5000
width=12)
                                       Filter: (SubPlan 1)
                                       SubPlan 1
                                         ->  Materialize  (cost=0.00..2.02 rows=900 width=4)
                                               ->  Streaming(type: BROADCAST)  (cost=0.00..1.52 rows=300
width=4)
                                                     Spawn on: All datanodes
                                                     ->  Seq Scan on t3  (cost=0.00..1.33 rows=100 width=4)
                           ->  Hash  (cost=323.30..323.30 rows=9999 width=4)
                                 ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..323.30 rows=10000
width=4)
                                       Spawn on: All datanodes
                                       ->  Seq Scan on t2  (cost=0.00..48.33 rows=10000 width=4)
(24 rows)
```

This is a typical distributed SQL plan, which is also the most common plan in distributed databases. The plan contains the main scan paths (index scan and sequential scan paths) and join path (Hash Left Join) mentioned above. In addition, **t1.a > ALL(SELECT a FROM t3)** generates a subplan that can be independently executed in the execution phase based on the constraints of table **t1**. The upper layer of the scan operator of table **t1** is a Stream(Redistribute) operator. In a distributed database, data in each table is distributed and stored on different DNs based on the distribution key. Therefore, when tables **t1** and **t2** are joined, data needs to be redistributed based on the equality condition of column **b** to ensure that data can be correctly joined on each DN. The upper layer of a Left Join operator is a Group By operator (HashAggregate), and then a Stream operator (Gather) for gathering the computing results returned by each DN on CNs. After the results are gathered, the Sort operator is executed, and finally the Limit operator is executed. As shown in the plan, the processing logic of the SQL execution plan is mapped to the original SQL statement, except for the Stream operator used for distributed computing.

- Distributed PGXC plan:

```
gaussdb=# SET enable_stream_operator = off;
SET
gaussdb=# EXPLAIN SELECT count(t1.a),t1.c FROM t1 LEFT JOIN t2 ON t1.b = t2.b WHERE t1.a >
ALL(SELECT a FROM t3) GROUP BY t1.c ORDER BY t1.c LIMIT 10;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
------
 Limit  (cost=250.11..250.12 rows=5 width=8)
   ->  Sort  (cost=250.11..250.12 rows=5 width=8)
         Sort Key: t1.c
         ->  HashAggregate  (cost=250.00..250.05 rows=5 width=8)
               Group By Key: t1.c
               ->  Hash Right Join  (cost=62.50..200.00 rows=10000 width=8)
                     Hash Cond: (t2.b = t1.b)
                     ->  Data Node Scan on t2 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=10000
width=4)
                           Node/s: All datanodes
                     ->  Hash  (cost=0.00..0.00 rows=5000 width=12)
                           ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=5000
width=12)
                                 Node/s: All datanodes
                                 Coordinator quals: (SubPlan 1)
                                 SubPlan 1
                                   ->  Materialize  (cost=0.00..0.50 rows=100 width=4)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                                ->  Data Node Scan on t3 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00
rows=100 width=4)
                                      Node/s: All datanodes
(17 rows)
```

The principle of a PGXC plan is to directly push down some simple SQL queries to DNs for computing and return the results to CNs for gathering. The degree of parallelism of a PGXC plan is lower than that of a Stream plan. Therefore, a PGXC plan is used only when a Stream plan cannot be generated. As shown in the preceding example, the GUC parameter **enable_stream_operator** is set to disable the Stream plan generation logic and forcibly select the PGXC plan. Compared with the Stream plan, the plan does not contain the Stream operator. In addition, the SeqScan operations on tables **t1**, **t2**, and **t3** are pushed down to DNs for execution. However, other operations, such as join, grouping, and sorting, are performed on the CNs.

- Distributed statement pushdown plan:

  In distributed scenarios, there is another type of query plan. That is, simple SQL statements are directly pushed down to each DN, the optimizer of each DN generates and executes the execution plan, and CNs only gather the results. A typical example is as follows:

  ```
  gaussdb=# EXPLAIN SELECT * from t1 WHERE b > 10;
                                  QUERY PLAN
  ------------------------------------------------------------------------------------
   Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=9982 width=16)
     Node/s: All datanodes
  (2 rows)
  ```

  The preceding SQL statement shows that the query statement logic of this type of plan is relatively simple. It does not require distributed computing of multi-table join data, and does not have complex restrictions such as aggregate functions, window functions, grouping, and sorting. Without depending on the additional computing capability of CNs, it can maximize the distributed parallel computing capability of nodes.

The statement for dropping the created tables in the preceding example is as follows:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# DROP TABLE t3;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

7.13.3 Operation Guide

Prerequisites

This section describes the basic plan selection process of the optimizer using the following table creation statement as an example:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# CREATE TABLE t2(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_a ON t2(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t2_b ON t2(b);
CREATE INDEX
gaussdb=# INSERT INTO t2 (a, b, c) VALUES (generate_series(1, 10000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 10000
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# SET explain_perf_mode=normal;
SET
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=on;
SET
```

Example

The following example uses different constant values of the filter condition of the same SQL statement to show the impact of the cost model on the plan selection of the CBO. Run the following SQL statement and check the selected plan (the filter condition value is **t2.b <= 100**):

```
gaussdb=# EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.a = t2.c AND t2.b <= 100;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=13.95..16.88 rows=100 width=32)
   Node/s: All datanodes
   ->  Merge Join  (cost=11.89..12.76 rows=100 width=32)
         Merge Cond: (t1.a = t2.c)
         ->  Index Scan using idx_t1_a on t1  (cost=0.00..44.47 rows=5000 width=16)
         ->  Sort  (cost=11.84..12.09 rows=100 width=16)
               Sort Key: t2.c
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..8.52 rows=100 width=16)
                     Spawn on: All datanodes
                     ->  Index Scan using idx_t2_b on t2  (cost=0.00..3.73 rows=100 width=16)
                           Index Cond: (b <= 100)
(11 rows)
```

When the filter condition is changed to **t2.b <= 3000**, the optimizer generates the following plan:

```
gaussdb=# EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.a = t2.c AND t2.b <= 3000;
                                QUERY PLAN
-----------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=49.51..299.97 rows=3000 width=32)
   Node/s: All datanodes
   ->  Hash Join  (cost=45.51..174.97 rows=3000 width=32)
         Hash Cond: (t2.c = t1.a)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..115.24 rows=3000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t2  (cost=0.00..28.84 rows=3000 width=16)
                     Filter: (b <= 3000)
         ->  Hash  (cost=24.67..24.67 rows=5001 width=16)
               ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
(10 rows)
```

In the preceding SQL statement, tables **t1** and **t2** are joined using INNER JOIN, and the join condition is **t1.a = t2.c AND t2.b <= 100**. The join condition **t1.a =**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

t2.c remains unchanged. When the constant value in **t2.b <=***?* is **100**, the merge join path is selected for execution, which uses the index conditions on **t1** and **t2** to improve the execution performance. However, when the constant value changes to **3000**, the execution plan changes to the hash join path. This indicates that the optimizer selects a proper query plan based on the joined tables and constraint changes, reducing manual optimization scenarios and improving efficiency in the actual production environment.

Follow-up Operations

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

7.13.4 Typical Issues

- Scenario 1: Why cannot an index scan path be generated?

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# SET enable_fast_query_shipping=off;
SET
gaussdb=# SET enable_stream_operator = on;
SET
gaussdb=# SET explain_perf_mode=normal;
SET
```

The table creation statement is shown above. Why cannot an index path be used when the following SQL statement is executed?

```
gaussdb=# EXPLAIN SELECT /*+indexscan(t1)*/* FROM t1;
WARNING:  unused hint: IndexScan(t1)
                          QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..233.05 rows=5000 width=16)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
(3 rows)

gaussdb=# EXPLAIN SELECT /*+indexscan(t1)*/* FROM t1 WHERE t1.b >= 100;
WARNING:  unused hint: IndexScan(t1)
                          QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.56..232.96 rows=4900 width=16)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=0.56..28.84 rows=4900 width=16)
         Filter: (b >= 100)
(4 rows)

gaussdb=# EXPLAIN SELECT /*+indexscan(t1)*/* FROM t1 ORDER BY c;
WARNING:  unused hint: IndexScan(t1)
                           QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=117.88..326.42 rows=5001 width=16)
   Merge Sort Key: c
   Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  ->  Sort  (cost=113.88..118.05 rows=5001 width=16)
        Sort Key: c
        ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
(6 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

The advantage of indexes lies in the ordered data and quick search for good filter conditions. Otherwise, in terms of random read/write, index scan does not have advantages over sequential scan. The preceding three SQL statements include neither filter conditions for accessing indexes nor sorting conditions on the index columns. Therefore, the index path is usually pruned when a scan path is generated to reduce the search space.

- Scenario 2: How to determine the selection of different join paths and join orders?

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# CREATE TABLE t2(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_a ON t2(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t2_b ON t2(b);
CREATE INDEX
gaussdb=# INSERT INTO t2 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# SET explain_perf_mode=normal;
SET
gaussdb=# SET enable_nestloop=on;
SET
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=on;
SET
```

Assume that the following SQL statement is used:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.b = t2.b;
```

The default execution plan selected by the optimizer is as follows:

```
gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.b = t2.b;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=187.03..576.53 rows=5000 width=32)
   Node/s: All datanodes
   ->  Hash Left Join  (cost=183.03..368.15 rows=5000 width=32)
         Hash Cond: (t1.b = t2.b)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      ->  Hash  (cost=162.20..162.20 rows=5001 width=16)
            ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                  Spawn on: All datanodes
                  ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)
```

Generally, you can use GUC parameters or hints to control the selection of a nested-loop join plan. You can use GUC parameters to increase the probability of selecting a nested-loop join plan only by disabling other path generation methods. However, you can use hints to specify the join type more accurately. The following are examples of these two methods:

```
--- Set GUC parameters to generate a nested-loop join plan.
gaussdb=# SET enable_nestloop=on;
SET
gaussdb=# SET enable_hashjoin=off;
SET
gaussdb=# SET enable_mergejoin=off;
SET
gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.b = t2.b;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..42220.27 rows=5000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..42011.90 rows=5000 width=32)
         Join Filter: (t1.b = t2.b)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..170.53 rows=5000 width=16)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)

-- Use hints to control the generation of the nested-loop join plan.
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=on;
SET
gaussdb=# EXPLAIN SELECT /*+nestloop(t1 t2)*/* FROM t1 LEFT JOIN t2 ON t1.b = t2.b;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..42220.27 rows=5000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..42011.90 rows=5000 width=32)
         Join Filter: (t1.b = t2.b)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..170.53 rows=5000 width=16)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)
```

Generally, the join order is selected based on the cost of a generated path. Some paths in certain join orders cannot be generated. However, if a path in a join order can be generated, you can use the leading hint to control the plan generation. The following uses the preceding SQL statements as an example to describe how to use the leading hint to specify table **t2** as the driving table and table **t1** as the driven table.

```
gaussdb=# SET enable_nestloop=off;
SET
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=off;
SET
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# EXPLAIN SELECT /*+leading((t2 t1))*/* FROM t1 LEFT JOIN t2 ON t1.b = t2.b;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=187.03..576.53 rows=5000 width=32)
   Node/s: All datanodes
   ->  Hash Right Join  (cost=183.03..368.15 rows=5000 width=32)
         Hash Cond: (t2.b = t1.b)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
               Spawn on: All datanodes
               ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
         ->  Hash  (cost=162.20..162.20 rows=5001 width=16)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..162.20 rows=5000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
(11 rows)
```

This plan uses the **leading((t2 t1))** hint to forcibly specify **t2** as the driving table and **t1** as the driven table. Run the following statements to drop tables **t1** and **t2**:

```
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

For details about how to use join order and join operation hints, see "Hint-based Optimization > Join Order Hints" and "Hint-based Optimization > Join Operation Hints" in *SQL Optimization*.

- Scenario 3: Why cannot a hash join path be generated?

  Use the following SQL statements as an example:

```
gaussdb=# CREATE TABLE t1(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a ON t1(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
gaussdb=# INSERT INTO t1 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# CREATE TABLE t2(a INT, b INT, c INT, d INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_a ON t2(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_t2_b ON t2(b);
CREATE INDEX
gaussdb=# INSERT INTO t2 (a, b, c) VALUES (generate_series(1, 5000), generate_series(1, 5000),
floor(random()*10));
INSERT 0 5000
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# SET explain_perf_mode=normal;
SET
gaussdb=# SET enable_nestloop=off;
SET
gaussdb=# SET enable_hashjoin=on;
SET
gaussdb=# SET enable_mergejoin=on;
SET
gaussdb=# EXPLAIN SELECT * FROM t1 LEFT JOIN t2 ON t1.b != t2.b;
                                QUERY PLAN
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
----------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..1166996.01 rows=24995000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..125537.63 rows=24995000 width=32)
         Join Filter: (t1.b <> t2.b)
         ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..500.46 rows=15000 width=16)
               ->  Streaming(type: BROADCAST)  (cost=0.00..475.46 rows=15000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(9 rows)

gaussdb=# EXPLAIN SELECT /*+hashjoin(t1 t2)*/* FROM t1 LEFT JOIN t2 ON t1.b != t2.b;
WARNING:  unused hint: HashJoin(t1 t2)
                                QUERY PLAN
----------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..1166996.01 rows=24995000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..125537.63 rows=24995000 width=32)
         Join Filter: (t1.b <> t2.b)
         ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..500.46 rows=15000 width=16)
               ->  Streaming(type: BROADCAST)  (cost=0.00..475.46 rows=15000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(9 rows)

gaussdb=# EXPLAIN SELECT /*+mergejoin(t1 t2)*/* FROM t1 LEFT JOIN t2 ON t1.b != t2.b;
WARNING:  unused hint: MergeJoin(t1 t2)
                                QUERY PLAN
----------------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..1166996.01 rows=24995000 width=32)
   Node/s: All datanodes
   ->  Nested Loop Left Join  (cost=0.00..125537.63 rows=24995000 width=32)
         Join Filter: (t1.b <> t2.b)
         ->  Seq Scan on t1  (cost=0.00..24.67 rows=5000 width=16)
         ->  Materialize  (cost=0.00..500.46 rows=15000 width=16)
               ->  Streaming(type: BROADCAST)  (cost=0.00..475.46 rows=15000 width=16)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..24.67 rows=5000 width=16)
(9 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# RESET explain_perf_mode;
RESET
```

The preceding SQL statements cannot determine whether a hash join or merge join plan is generated even through hints or GUC parameters. Except nested-loop join, the generation of hash join and merge join paths has some restrictions. The hash join algorithm is based on hash tables. For non-equi-join, target data cannot be quickly found using hash tables, and therefore, no hash join path can be generated.

7.14 AutoHint for Intelligent Hint Recommendation

Overview

AutoHint automatically identifies operators that take a long time in the execution plan tree of slow SQL statements, generates candidate hints for replacement, explores the hints using the greedy policy, and finally generates a hint combination that can improve the execution performance of the current slow SQL

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

statements. This function can improve the performance of slow SQL statements and reduce the labor and time costs.

NOTICE

AutoHint is not supported in standby clusters for DR.

Prerequisites

- The database is working properly.
- You have the **super** permission.
- If cardinality estimation feedback is disabled and adaptive cost estimation are enabled, you need to run multiple queries in advance to ensure that the cardinality estimation feedback model and cost estimation model have converged.

Usage Guide

Run related queries by referring to the AutoHint syntax to obtain the hint recommendation result.

Best Practices

In the scenario where cardinality estimation feedback is disabled and adaptive cost estimation are disabled (by setting **enable_adaptive_cost** to **off** and **adaptive_cardest_strategy** to **use_statistics**):

Step 1 Start the database and import data.

Step 2 Execute slow SQL statements, for example, query SQL statements that contain multiple join paths and join types, and observe the execution time and execution tree in the default scenario. The **query11d** statement in the **JOB** data set is used as an example.

```
gaussdb=# EXPLAIN ANALYZE SELECT MIN(cn.name) AS from_company,
       MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_based_on_book
FROM company_name AS cn,
     company_type AS ct,
     keyword AS k,
     link_type AS lt,
     movie_companies AS mc,
     movie_keyword AS mk,
     movie_link AS ml,
     title AS t
WHERE cn.country_code !='[pl]'
  AND ct.kind != 'production companies'
  AND ct.kind IS NOT NULL
  AND k.keyword IN ('sequel',
                    'revenge',
                    'based-on-novel')
  AND mc.note IS NOT NULL
  AND t.production_year > 1965
  AND lt.id = ml.link_type_id
  AND ml.movie_id = t.id
  AND t.id = mk.movie_id
  AND mk.keyword_id = k.id
  AND t.id = mc.movie_id
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 AND mc.company_type_id = ct.id
 AND mc.company_id = cn.id
 AND ml.movie_id = mk.movie_id
 AND ml.movie_id = mc.movie_id
 AND mk.movie_id = mc.movie_id;
```

Step 3 Run AutoHint commands to recommend hints.

```
gaussdb=# AUTOHINT ANALYZE
SELECT MIN(cn.name) AS from_company,
       MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_based_on_book
FROM company_name AS cn,
     company_type AS ct,
     keyword AS k,
     link_type AS lt,
     movie_companies AS mc,
     movie_keyword AS mk,
     movie_link AS ml,
     title AS t
WHERE cn.country_code !='[pl]'
  AND ct.kind != 'production companies'
  AND ct.kind IS NOT NULL
  AND k.keyword IN ('sequel',
                    'revenge',
                    'based-on-novel')
  AND mc.note IS NOT NULL
  AND t.production_year > 1965
  AND lt.id = ml.link_type_id
  AND ml.movie_id = t.id
  AND t.id = mk.movie_id
  AND mk.keyword_id = k.id
  AND t.id = mc.movie_id
  AND mc.company_type_id = ct.id
  AND mc.company_id = cn.id
  AND ml.movie_id = mk.movie_id
  AND ml.movie_id = mc.movie_id
  AND mk.movie_id = mc.movie_id;
```

Step 4 View the recommended hints and the expected improvement effect.

```
                AUTOHINT
-------------------------------------------
 Recommended hintset:
   MergeJoin(@sel$1 k@sel$1 mk@sel$1 mc@sel$1)

 Expected boost:
   planning:     -7%
   execution:    62%
   overall:      60%
```

Step 5 Use the recommended hints to check whether the execution plan changes and whether the execution performance is improved.

```
gaussdb=# EXPLAIN ANALYZE SELECT /*+MergeJoin(@sel$1 k@sel$1 mk@sel$1 mc@sel$1)*/ MIN(cn.name)
AS from_company,
       MIN(mc.note) AS production_note,
       MIN(t.title) AS movie_based_on_book
FROM company_name AS cn,
     company_type AS ct,
     keyword AS k,
     link_type AS lt,
     movie_companies AS mc,
     movie_keyword AS mk,
     movie_link AS ml,
     title AS t
WHERE cn.country_code !='[pl]'
  AND ct.kind != 'production companies'
  AND ct.kind IS NOT NULL
  AND k.keyword IN ('sequel',
                    'revenge',
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                'based-on-novel')
AND mc.note IS NOT NULL
AND t.production_year > 1965
AND lt.id = ml.link_type_id
AND ml.movie_id = t.id
AND t.id = mk.movie_id
AND mk.keyword_id = k.id
AND t.id = mc.movie_id
AND mc.company_type_id = ct.id
AND mc.company_id = cn.id
AND ml.movie_id = mk.movie_id
AND ml.movie_id = mc.movie_id
AND mk.movie_id = mc.movie_id;
```

----End

In the scenario where cardinality estimation feedback and adaptive cost estimation are enabled:

After data is imported, execute slow SQL statements for multiple times until the SQL execution plan is stable. This indicates that the cardinality estimation feedback model and cost estimation model have converged. The subsequent operations are the same as the preceding steps.

Troubleshooting

In case that the exploration process fails due to network problems or the maximum number of connections, change the upper limit of the maximum number of connections or wait and use this functionality again until the resources are idle.

7.15 DSL Rule Engine

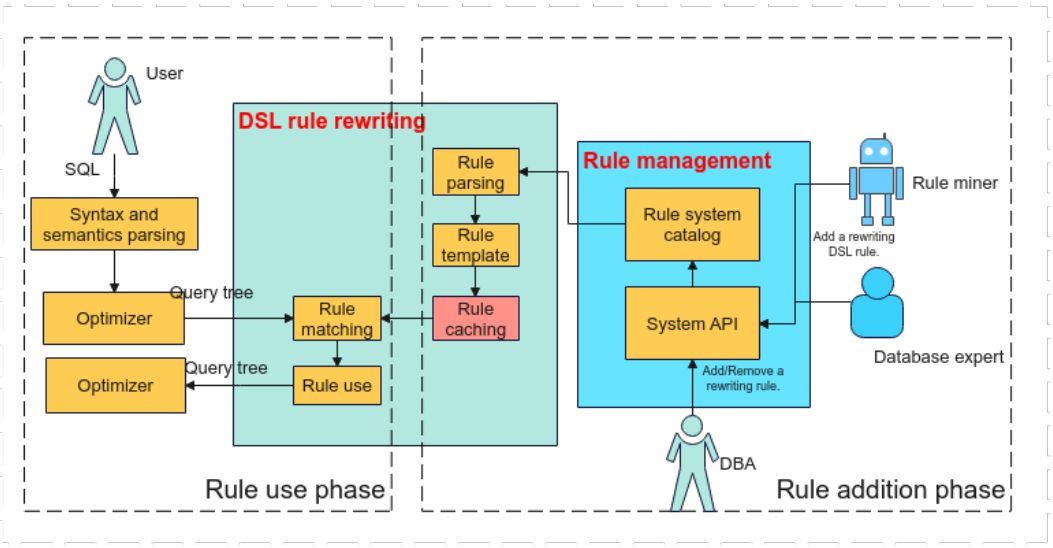
7.15.1 Overall Process of the DSL Rule Engine

The DSL rule engine describes SQL query rewriting rules by using a domain-specific language (DSL). It develops specific syntaxes through unified atomic APIs. By flexibly orchestrating these syntaxes and leveraging their symbolic description, query rewriting rules can be integrated as plug-ins and dynamically loaded, enhancing the ease of development, deployment, and flexible scalability of query rewriting.

The DSL rule engine is embedded in the SQL engine and rewrites SQL statements in the running state by loading DSL rules, helping the database optimizer to generate better execution paths and improving execution performance. **Figure 7-33** shows the overall process of the DSL rule engine.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-33 Overall process of the DSL rule engine



As shown in the preceding figure, the implementation of the DSL rule engine includes two phases:

In the rule addition phase, maintenance personnel add new DSL rewriting rules through the DSL rule management module and synchronize the content in the system catalog to the global cache pool of the DSL rule engine through the RELOAD mechanism; in the rule use phase, the DSL rule engine loads these rules from the cache pool in the application framework and traverses the rule set based on the priority policy to complete matching and rewriting.

- Rule addition phase:
  - DSL rules are classified into two types: built-in rules and external rules. Built-in rules are preset in the system catalog of DSL rule management released with the database by kernel developers, as shown in **Table 7-18**. Built-in rules cannot be deleted but can be disabled. External rules are compiled by DBAs using a DSL and their matching and replacement in abnormal scenarios is ensured by the rule designers. The system does not guarantee the equivalence of external rules. For these external DSL rule sets, the engine only ensures the correctness of the scheduling logic of the framework.
  - After a new DSL rule is added, it can be used by performing the RELOAD operation. Pay attention to the lifecycle changes of DSL rule sets when they are running in the DSL rule engine. For new SQL session requests, use the updated DSL rule sets to schedule the rewriting process. For sessions that are in the rewriting process, use the old copies to complete the entire query rewriting process.
- Rule use phase, including:
  a. A user enters an SQL request whose syntax and semantic are then parsed by the database to obtain a query tree.
  b. The query tree obtained in step 1 is used as the input parameter to enter the rewriting module of the DSL rule engine. The rewriting module is located in the query rewriting rule model of the optimizer and the rule application includes the following sub-processes:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

i. DSL rule matching and application: The nodes of the query tree are matched with the template and the matched nodes are rewritten equivalently based on rules to generate the target query tree.

ii. DSL rule filtering: The DSL rule engine filters rules in a rule set based on the rule status, sorts rules based on the rule priority, and schedules each DSL rule to execute the process in **the previous step** on the query tree.

iii. Rewriting termination: The DSL rule engine terminates the rewriting when: (1) A rule set is completely scheduled twice and no rewriting occurs; or (2) the number of times that a rule set is scheduled reaches the threshold set by the system.

Table 7-18 Built-in rules

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 1: Join elimination	• Sub-scenario 1.1: Elimination of outer join with the DISTINCT keyword rulename:
	distinct_enabled_outer_join_elimination
	• Sub-scenario 1.2: Elimination of outer join with the GROUP BY condition rulename:
	group_by_outer_join_elimination
		• Sub-scenario 1.1: For a left outer join or right outer join, if the projection column contains only the column values from the outer table and has a DISTINCT constraint, the outer join can be eliminated.
		The rule is enabled by default.
		• Sub-scenario 1.2: If the GROUP BY condition meets the deduplication semantics of DISTINCT, the outer join can also be eliminated.
		The rule is enabled by default.
			• Sub-scenario 1.1:
			`SELECT DISTINCT t1.* FROM t1 LEFT JOIN t2 ON t1.c1 = t2.c1;`
			It is rewritten equivalently as follows:
			`SELECT DISTINCT t1.* FROM t1;`
			• Sub-scenario 1.2:
			`SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.c = t2.c GROUP BY t1.a, t1.b, t1.c;`
			It is rewritten equivalently as follows:
			`SELECT t1.* FROM t1;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

	Sub-scenario 2: Elimination of outer join with the "always false" condition
	rulename:
	always_false_outer_join_elimination
		• For a left outer join or right outer join, if the JOIN condition is always false, no row of the outer table matches rows of the inner table. As a result, the projection column returns each row of the outer table and pads columns with **NULL** for the inner table. Therefore, the outer join can be eliminated.
		• The rule is enabled by default.
			Assume that tables **T1** and **T2** each have only one attribute column.
			`SELECT * FROM t1 LEFT JOIN t2 ON 1 = 0;`
			It is rewritten equivalently as follows:
			`SELECT t1.c1, NULL FROM t1;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

	Sub-scenario 3: Elimination of semi join on the same column in the same table
	rulename:
	same_table_semi_join_elimination
		• As shown in the example for this scenario, if the main query and subquery reference the same table **t1** and the join condition is the same column, **t1** must be a simple table. According to the semantics of the EXISTS subquery, if **t1.c1** is not all **NULL**, the rows where **t1.c1** is not **NULL** can be returned after the EXISTS subquery is filtered. If **t1.c1** is all **NULL**, the EXISTS subquery returns **FALSE**. In this case, the result set is empty. In this scenario, some equivalent optimizations can be performed to eliminate the semi join.
		• The rule is enabled by default.
			`SELECT * FROM t1 WHERE EXISTS(SELECT 1 FROM t1 tt WHERE t1.c1 = tt.c1);`
			It is rewritten equivalently as follows:
			`SELECT * FROM t1 WHERE t1.c1 IS NOT NULL;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 2: Pushdown of the LIMIT condition	• Sub-scenario 1.1: Pushdown of LIMIT in outer join to a subquery rulename:
	left_join_branch_limit_pushdown
	• Sub-scenario 1.2: Pushdown of LIMIT in cross join to a subquery rulename:
	cross_join_branch_limit_pushdown
		• If the outer join or cross join does not contain WINDOW FUNCTION, DISTINCT, GROUP BY, or HAVING, and the WHERE or ORDER BY condition is related to only one of the tables to be joined, the LIMIT condition can be pushed down to one side (for an outer join) or multiple sides (for a multi-table cross join) of the joined tables. Pushing down LIMIT can effectively reduce the number of joined rows, thereby reducing the performance overhead of plan execution.
		• The rule is enabled by default.
			• Sub-scenario 1.1:
			`SELECT * FROM t1 LEFT JOIN t2 ON t1.c1 = t2.c1 LIMIT 1;`
			It is rewritten equivalently as follows:
			`SELECT * FROM (SELECT * FROM t1 LIMIT 1) t LEFT JOIN t2 ON t.c1 = t2.c1 LIMIT 1;`
			• Sub-scenario 1.2:
			`SELECT 1 FROM t1, t2 WHERE t1.c1 > 0 ORDER BY t1.c1 LIMIT 1;`
			It is rewritten equivalently as follows:
			`SELECT 1 FROM (SELECT 1 FROM t1 WHERE t1.c1 > 0 ORDER BY t1.c1 LIMIT 1) tm1, (SELECT 1 FROM t2 LIMIT 1) tm2 LIMIT 1;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 3: Elimination of redundant DISTINCT	• Sub-scenario 1.1: The DISTINCT constraint can be eliminated when all projection columns are constants. rulename: const_only_distinct_elimination_base
	• Sub-scenario 1.2: The DISTINCT constraint can be eliminated when all projection columns are constants and there is an **OFFSET** with a value greater than 0. The entire SQL statement can be simplified to a restrictive condition of FALSE. rulename: const_only_distinct_elimination_limit
	• Sub-scenario 1.3: The GROUP BY constraint can be
		• Sub-scenario 1.1: If the projection columns on which SELECT is performed contain only constants, you can add the LIMIT 1 constraint to eliminate DISTINCT and end the data traversal in advance. The rule is enabled by default.
		• Sub-scenario 1.2: If the projection columns on which SELECT is performed contain only constants and there is an **OFFSET** with a value greater than 0, the DISTINCT constraint can be eliminated. The entire SQL statement can be simplified to a restrictive condition of FALSE, and the SQL statement execution can be ended in advance. The rule is disabled by default.
		• Sub-scenario 1.3: If the projection columns on
			• Sub-scenario 1.1:
			`SELECT DISTINCT 1,2 FROM t1;`
			It is rewritten equivalently as follows:
			`SELECT 1,2 FROM t1 LIMIT 1;`
			• Sub-scenario 1.2:
			`SELECT DISTINCT 1,2 FROM t1 OFFSET 1;`
			It is rewritten equivalently as follows:
			`SELECT 1,2 FROM t1 WHERE FALSE;`
			• Sub-scenario 1.3:
			`SELECT 1,2 FROM t1 GROUP BY 1,2;`
			It is rewritten equivalently as follows:
			`SELECT 1,2 FROM t1 LIMIT 1;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
	eliminated when all projection columns are constants and the GROUP BY constraint includes all projection columns. rulename: const_only_distinct_elimination_groupby	which SELECT is performed contain only constants, the GROUP BY constraint contains all projection columns, and there is no HAVING condition or GROUPING SET, you can eliminate GROUP BY and add the LIMIT 1 constraint to end the data traversal in advance. The rule is disabled by default.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

	• Sub-scenario 2.1: The DISTINCT constraint can be eliminated when the projection columns contain the UNIQUE constraint. rulename:
	unique_key_distinct_elimination_base
	• Sub-scenario 2.2: GROUP BY can be eliminated when the projection columns contain the UNIQUE constraint. rulename:
	unique_key_distinct_elimination_groupby
		• Sub-scenario 2.1: If the selected projection columns have a UNIQUE constraint and no complex tables are specified in the FROM clause, DISTINCT can be eliminated. Assume that table **t1** has a UNIQUE NOT NULL constraint on column **c1**.
		The rule is enabled by default.
		• Sub-scenario 2.2: If the projection columns selected by GROUP BY have a UNIQUE constraint, the constraint is equivalent to the DISTINCT constraint in terms of semantics. Therefore, GROUP BY can also be eliminated in this scenario.
		The rule is disabled by default.
			• Sub-scenario 2.1:
			`SELECT DISTINCT c1, c2, c3 FROM t1;`
			It is rewritten equivalently as follows:
			`SELECT c1, c2, c3 FROM t1;`
			• Sub-scenario 2.2:
			`SELECT t1.a, t1.b, t1.c FROM t1 GROUP BY t1.a;`
			It is rewritten equivalently as follows:
			`SELECT t1.a, t1.b, t1.c FROM t1;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
Scenario 4: Complex cases	Sub-scenario 1: Acceleration of outer join in a stream plan by using semi join rulename: semi_join_accelerated_outer_join	• When a small table is left joined with a large table, if the join condition is not the distribution key, the large table must be redistributed and the small table cannot be broadcast. As a result, the performance is poor. In this case, you can perform a semi join between the large table and the small table to filter out most data before performing an outer join with the small table. In this way, the small table can be broadcast. • The rule is disabled by default.	SELECT * FROM t2 small LEFT JOIN t1 big ON small.c1 = big.c1; It is rewritten equivalently as follows: SELECT * FROM t2 small LEFT JOIN (SELECT * FROM t1 big WHERE big.c1 IN (SELECT c1 FROM t2)) fakebig ON small.c1 = fakebig.c1;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
	Sub-scenario 2: Elimination of expression sublinks when they are compared and one is a substructure of another rulename: subLink_compare_elimination	• When two expression sublinks containing the Count aggregate function are compared for equality, if one sublink can be expressed as a substructure of the other sublink and the filter conditions differ only in an EXISTS sublink condition, optimization can be performed. The purpose is to determine the cases where **FALSE** is returned for the EXISTS condition. In this case, whether the values of two COUNT functions equal depends on the **TRUE** result of other conditions When other conditions are **TRUE**, the filter conditions of the two sublinks are opposite according to the semantics of the AND condition, and the COUNT() values must be different. Otherwise, the values are the same. Therefore,	SELECT t1.a, t1.b FROM t1,t2 WHERE t1.a = t2.a AND t1.b = t1.b AND (SELECT COUNT(*) FROM t1 as t11 WHERE t11.a = t1.a AND t11.b = t1.b) = (SELECT COUNT(*) FROM t1 AS t21 WHERE t21.a = t1.a AND t21.b = t1.b AND EXISTS(SELECT 1 FROM t3 WHERE t1.a = t3.a)); It is rewritten equivalently as follows: SELECT t1.a, t1.b FROM t1,t2 WHERE t1.a = t2.a AND t1.b = t1.b AND NOT EXISTS (SELECT 1 FROM t1 as t11 WHERE t11.a = t1.a AND t11.b = t1.b AND NOT EXISTS(SELECT 1 FROM t3 WHERE t1.a = t3.a));

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
		in this scenario, NOT EXISTS can be used for equivalent rewriting to reduce the number of sublinks. • The rule is enabled by default.	

	• Sub-scenario 3.1: Pushdown of constant predicates to the projection columns of a subquery rulename:
	const_predicate_pushdown_to_project
	• Sub-scenario 3.2: Pushdown of constant predicates to the filter condition of a subquery rulename:
	const_predicate_pushdown_to_filter
		• If the main query contains constant predicate conditions, that is, one side of the expression is a table column and the other side is a constant, and the table column is also found in the query condition of the sublink, the constant condition can be pushed down to the sublink.
		• The rule is disabled by default.
			• Sub-scenario 3.1:
			`SELECT t1.a, (SELECT sum(t2.a) FROM t2 WHERE t2.b = t1.b) FROM t1 WHERE t1.b = 1;`
			It is rewritten equivalently as follows:
			`SELECT t1.a, (SELECT sum(t2.a) FROM t2 WHERE t2.b = t1.b AND t1.b = 1) FROM t1 WHERE t1.b = 1;`
			• Sub-scenario 3.2:
			`SELECT t1.a FROM t1 WHERE t1.b = 1 AND (SELECT SUM(t2.a) FROM t2 WHERE t2.b = t1.b) = 0;`
			It is rewritten equivalently as follows:
			`SELECT a FROM t1 WHERE b = 1 AND (SELECT sum(t2.a) FROM t2 WHERE t2.b = t1.b AND t1.b = 1) = 0;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
	Sub-scenario 4: Conversion of a correlated expression sublink to a cross join rulename: subLink_to_crossJoin	• For an equality expression sublink with an aggregate function, if the sublink is compared with a constant, the sublink can be extracted as a CTE table and cross joined with the main query. In specific data distribution scenarios in distributed mode, STREAM BROADCAST can be used to improve performance. • The rule is disabled by default.	SELECT * FROM t1 WHERE t1.a = 1 AND (SELECT count(*) FROM t2 WHERE t2.b = t1.b AND t2.b = 1) = 3; It is rewritten equivalently as follows: WITH cte AS (SELECT count(*) AS c, b FROM t2 WHERE t2.b = 1 GROUP BY b) SELECT * FROM t1, cte WHERE t1.a = 1 AND cte.b = t1.b AND cte.c = 3;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

	Sub-scenario 5: Elimination of aggregation when the expression sublink contains aggregate function Count and is compared with 0
	rulename:
	count_zero_elimination
		• For a correlated expression sublink, if the projection columns in the sublink contain the COUNT() function and is compared with 0 for equality, **TRUE** is returned only when the filter condition is not met. Therefore, it can be equivalently rewritten to a left join, and the aggregate function and subplan can be eliminated by filtering IS NULL columns in the inner table.
		• The rule is enabled by default.
			`SELECT t1.* FROM t1 WHERE t1.a = 1 AND (SELECT count(*) FROM t2 WHERE t2.b = t1.b AND t2.b = 1) = 0;`
			It is rewritten equivalently as follows:
			`SELECT t1.* FROM t1 LEFT JOIN t2 ON t2.b = 1 AND t2.b = t1.b WHERE t1.a = 1 AND t2.b IS NULL;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
Scenario 5: Self-join elimination based on the UNIQUE constraint of the inner join	To eliminate self-joins based on the UNIQUE constraint of the DSL rule engine, the following two rules are required: • Rule 1: Convert the inner join query to a cross join. rulename: inner_join_to_cross_join • Rule 2: Eliminate self-joins based on the UNIQUE constraint of the cross join. rulename: self_crossjoin_elimination	• When inner join of multiple tables in a query block involves PRIMARY KEY or NOT NULL UNIQUE constraints, redundant join operations can be eliminated through query rewriting rules. This reduces unnecessary join operations in the query and improves the overall query performance. The principle is that the inner join of the same table has a strict one-to-one matching relationship in terms of the PRIMARY KEY or NOT NULL UNIQUE constraint. Therefore, even if the result set has redundant data, all data can be obtained by scanning the original table once. • The rule is enabled by default.	SELECT * FROM t1 tt1 INNER JOIN t1 tt2 ON tt1.a = tt2.a INNER JOIN t1 tt3 ON tt1.a = tt3.a WHERE tt1.b > 100; It is rewritten equivalently as follows: SELECT tt1.*, tt1.*,tt1.* FROM t1 tt1 WHERE tt1.b > 100 AND tt1.a IS NOT NULL;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 6: Rewriting ALL non-correlated sublinks	• Sub-scenario 1.1: ALL non-correlated sublinks in the filter condition can be rewritten as expression sublinks of the MAX aggregate function. rulename: allsublink_add_max_filter
	• Sub-scenario 1.2: ALL non-correlated sublinks in the filter condition can be rewritten as expression sublinks of the MIN aggregate function. rulename: allsublink_add_min_filter
	• Sub-scenario 1.3: ALL non-correlated sublinks in the JOIN condition can be rewritten as expression sublinks of the MAX
		• A typical SQL statement is as follows:
		`SELECT a FROM t1 WHERE a > ALL(SELECT a FROM t2);`
		The preceding SQL query indicates that table **t1** is traversed and rows that meet the filter condition are returned. That is, all values in the column **a** in table **t1** are greater than those in the column **a** in table **t2**. According to the equivalent logic, the condition is equivalent to ensuring that the maximum value of column **a** in table **t1** is greater than that of the column **a** in table **t2**. For non-correlated sublink comparison scenarios of the >ALL, >= ALL, < ALL, and <= ALL types in the filter condition or JOIN condition, the performance can be optimized through equivalent rewriting.
			• Sub-scenario 1.1:
			`SELECT a FROM t1 WHERE a > ALL(SELECT a FROM t2);`
			It is rewritten equivalently as follows:
			`SELECT a FROM t1 WHERE CASE WHEN ((SELECT max(t2.a) AS a FROM t2)) IS NULL THEN true ELSE a > ((SELECT max(t2.a) AS a FROM t2)) END;`
			• Sub-scenario 1.2:
			`SELECT a FROM t1 WHERE a < ALL(SELECT a FROM t2);`
			It is rewritten equivalently as follows:
			`SELECT a FROM t1 WHERE CASE WHEN ((SELECT min(t2.a) AS a FROM t2)) IS NULL THEN true ELSE a < ((SELECT min(t2.a) AS a FROM t2)) END;`
			• Sub-scenario 1.3:
			`SELECT * FROM t1 LEFT JOIN t4 ON t1.b = t4.b AND t1.a > ALL(SELECT a FROM t2);`
			It is rewritten equivalently as follows:
			`SELECT * FROM t1 LEFT JOIN t4 ON t1.b = t4.b AND CASE WHEN ((SELECT max(t2.a) AS a FROM t2)) IS NULL THEN true ELSE t1.a > ((SELECT max(t2.a) AS a FROM t2)) END;`
			• Sub-scenario 1.4:
			`SELECT * FROM t1 LEFT JOIN t4 ON t1.b = t4.b AND t1.a < ALL(SELECT a FROM t2);`
			It is rewritten equivalently as follows:
			`SELECT * FROM t1 LEFT JOIN t4 ON t1.b = t4.b AND CASE WHEN ((SELECT min(t2.a) AS a FROM t2)) IS NULL THEN true ELSE t1.a < ((SELECT min(t2.a) AS a FROM t2)) END;`
			NOTE
			The optimization rules of the >ALL sublinks in examples 1.1 and 1.3 also apply to the >= connection scenario. Similarly, the optimization rules of the <ALL sublinks in examples 1.2 and 1.4 also apply to the <= connection scenario.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
	aggregate function. rulename: allsublink_add_max_join • Sub-scenario 1.4: ALL non-correlated sublinks in the JOIN condition can be rewritten as expression sublinks of the MIN aggregate function. rulename: allsublink_add_min_join	• The rule is disabled by default.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 7: Filter condition of MIN/MAX predicate pushdown from HAVING	Sub-scenario: Supporting filter condition of MIN/MAX predicate pushdown from HAVING
	rulename:
	minmax_predicate_pushdown
		• In some scenarios, predicates in the HAVING condition can be pushed down to the WHERE condition to filter data in advance while ensuring result consistency. This optimizes the execution performance. One type of scenarios is that the HAVING condition contains the MIN and MAX comparison predicates. The following is a typical SQL statement:
		`SELECT a, min(b) FROM t1 GROUP BY a HAVING min(b) < 10;`
		According to the preceding SQL query, the objective is to filter out the aggregation result after grouping the data meeting the **min(b)<10** condition. If the rows where **b** is less than 10 can be filtered out before grouping, the amount of data to be calculated can be reduced. Based on this
			• Example 1:
			`SELECT a, min(b) FROM t1 GROUP BY a HAVING min(b) < 10;`
			It is rewritten equivalently as follows:
			`SELECT a, min(b) FROM t1 WHERE b < 10 GROUP BY a HAVING min(b) < 10;`
			• Example 2:
			`SELECT a, max(b) FROM t1 GROUP BY a HAVING max(b) > 10;`
			It is rewritten equivalently as follows:
			`SELECT a, max(b) FROM t1 WHERE b > 10 GROUP BY a HAVING max(b) > 10;`
			NOTE
			The optimization rule in example 1 is using < for connecting the HAVING predicates, and it also applies when <= is used. Similarly, in example 2, the optimization rule is using > for connecting the HAVING predicates, and it also applies when >= is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
		logic, a new predicate **b < 10** can be deduced and pushed down to the filter condition. • The rule is enabled by default.	

Scenario	Sub-scenario	Description	Example SQL Rewriting

Scenario 8: Pushing down LIMIT to UNION ALL subqueries	Sub-scenario: Pushing down LIMIT to UNION ALL subqueries
	rulename:
	unionall_limit_pushdown
		• In the standard SQL syntax, UNION ALL combines the result sets of two or more SELECT subqueries, and does not sort or deduplicate the results. LIMIT limits the number of returned final data sets. If both the ORDER BY and LIMIT conditions are pushed down to the subqueries, the ordered nature of indexes can be used to eliminate the need for sorting, and the LIMIT condition can be used to terminate the computation of each subquery in advance. In this way, the main query only needs to perform a secondary sort on a small amount of data and then retrieve the first *n* results (*n* is specified by LIMIT). The following is a typical SQL example:
		`SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2) ORDER BY a1 LIMIT 2;`
			`SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2) LIMIT 2;`
			It is rewritten equivalently as follows:
			`SELECT * FROM ((SELECT * FROM t1 LIMIT 2) UNION ALL (SELECT * FROM t2 LIMIT 2)) LIMIT 2;`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
		• By default, this rule is disabled for upgrade scenarios and enabled for installation scenarios.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Sub-scenario	Description	Example SQL Rewriting
Scenario 9: Pushing down LIMIT to subqueries	Sub-scenario: Pushing down LIMIT to subqueries rulename: subquery_limit_pushdown	• Generally, SQL statements are executed in the following sequence: FROM -> JOIN ON -> WHERE -> GROUP BY -> HAVING -> SELECT -> DISTINCT -> ORDER BY -> LIMIT When SELECT is executed, each data record in the previous result set needs to be calculated. If complex calculations such as sublinks are involved, the overall execution time increases exponentially. According to the preceding execution sequence, the sorting and LIMIT operations are usually performed at the end. From the perspective of result equivalence, filtering redundant result sets before sublink calculation on valid data ensures the correctness of execution results and improves execution	SELECT (SELECT min(a) FROM t2 WHERE t2.c >= t1.b) AS a1 FROM t1 INNER JOIN t3 ON t1.a = t3.a ORDER BY t1.a LIMIT 10; It is rewritten equivalently as follows: SELECT (SELECT min(a) FROM t2 WHERE t2.c >= t.b) AS a1 FROM (SELECT t1.* FROM t1 INNER JOIN t3 ON t1.a = t3.a ORDER BY t1.a LIMIT 10) t ORDER BY t.a LIMIT 10;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

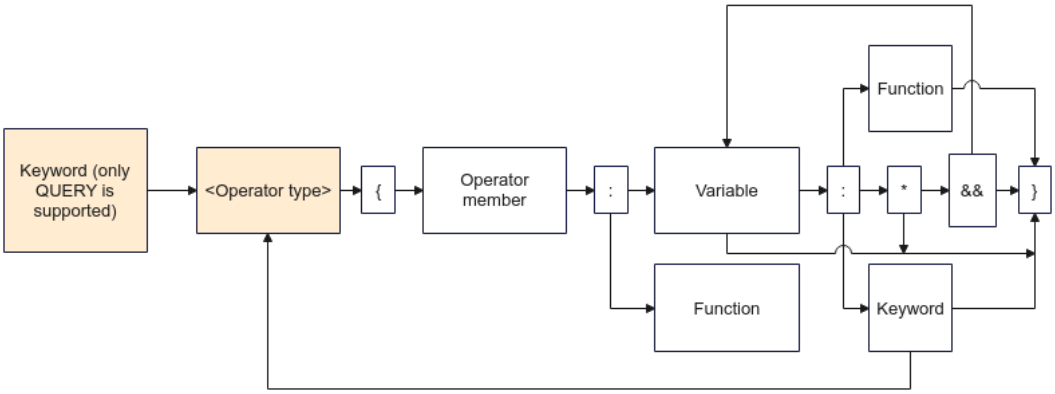
Scenario	Sub-scenario	Description	Example SQL Rewriting
		performance. The following is a typical SQL example: `SELECT (SELECT min(a) FROM t2 WHERE t2.c >= t1.b) AS a1 from t1 INNER JOIN t3 ON t1.a = t3.a ORDER BY t1.a LIMIT 10;` • By default, this rule is disabled for upgrade scenarios and enabled for installation scenarios.	

7.15.2 DSL Rule Writing Specifications

7.15.2.1 DSL Rule Syntax

The DSL rule engine defines a set of syntax specifications to rewrite matched nodes in the query tree according to a template. The syntax is composed of keywords, symbols, variables, functions, and constraints. **Figure 7-34** shows the syntax composition.

Figure 7-34 Syntax



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-19 DSL rule syntax elements

Element	Description	Limitation
Keyword	• Keyword: corresponds to a part of the query tree. For example, the FILTER keyword indicates the constraint in the query tree. • Angle brackets (<>): specify the keyword type. For example, JOIN<INNER/LEFT> specifies various JOIN types. If it is left empty, the default type will be used. For details about the default type, see the supported types in **Table 7-20** and **Table 7-21**. • Each operator is enclosed in braces ({}). The content in braces indicates the entire syntax keyword. Each keyword has its own variables, which correspond to the members in the query tree expression. For example, **JOIN<INNER>{LEFT: RIGHT: ON: }** indicates an INNER JOIN plan, where **LEFT** indicates the left subtree of JOIN, **RIGHT** indicates the right subtree of JOIN, and **ON** indicates the JOIN condition.	• The first keyword can only be QUERY. • The keyword type cannot be left empty if no default type is defined; otherwise, a parsing error will be reported.
Symbol	• Colon (:): symbol for explanation, which assigns a value to an operator member or variable. • AND (&&): means that all conditions must be met. • Wildcard character (*): indicates fuzzy match.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

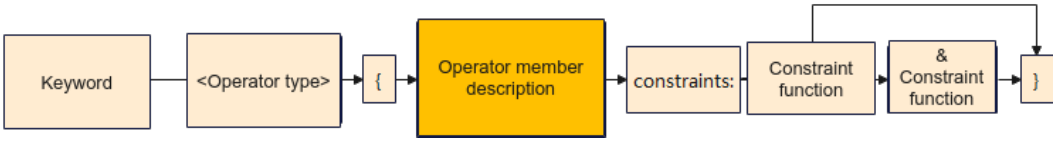
Element	Description	Limitation

Variable	Variables, represented by the DSL in the form of '$' + var_string, match expressions in the query tree and form mapping relationships for subsequent processing. The following uses the JOIN keyword as an example:
	`JOIN<INNER>{`
	`  LEFT: $l_var: *`
	`  RIGHT: $r_val: *`
	`  ON: $cond: *`
	`}`
	Where, **LEFT:** *$l_var*: *indicates the left subtree of JOIN, the variable *$l_var* matches all expressions of the left subtree to form a mapping relationship with the right subtree expressions matched with the variable *$r_val*, and *$cond* indicates the JOIN conditions.
		-

Function	Functions are the basis of rule implementation. Complex semantics such as constraints need to be represented by functions. For details, see **Table 7-23** and **Table 7-24**. • Constraint functions check whether the matched expressions meet the specified conditions. • Expression calculation functions use variables as inputs, output calculation results, and assign them to newly declared variables for recording.	• Functions together with their input parameters must be separated by parentheses (()); otherwise, a parsing error will be reported. • The variables serving as the function input parameters must be mapped and assigned; otherwise, an error will be reported when the variables are used.
Limitation	Constraints check whether specified conditions are met. When there are multiple constraints, the constraint functions can be combined by using AND (&&), as shown in **Figure 7-35**. If all constraints are met, the matching is successful. Each operator can have its own constraints. Constraints are generally composed of constraint functions. For details about constraint functions, see **Table 7-23**.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-35 Constraint representation



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-20 SQL syntax keywords

Keyword	Syntax	Type	Description

QUERY	QUERY<QUERY_TYPE>
	{
	PREPROCESS: ($PREPROCESS)
	CTE: ($cte_expr: *)
	PROJECT: ($proj_expr: *)
	FROM: ($from_expr: *)
	FILTER: ($filter_expr: *)
	GROUP: GROUP<> {$group_expr: *}
	SORT: ($sort_expr: *)
	LIMIT: LIMIT<> {$limit_expr: *}
	POSTPROCESS: ($POSTPROCESS)
	MAINBLOCK: false
	}
		• QUERY_TYPE: supported SQL command type. The options are as follows: **SELECT**, **DELETE**, **UPDATE**, and **INSERT**.
		• If the keyword type is not explicitly specified, the default type is used, which matches all supported DML types. When keywords are used for matching, multiple types can be combined. Different types are separated by vertical bars (|), for example, SELECT|DELETE.
			This keyword is used to match a complete QUERY or SUBQUERY structure. The inherent parameters are as follows (the sequence of the following parameters is fixed. Note that all parameters support omission, indicating that the corresponding parameters are not matched or the corresponding operations are not performed.):
			• **PREPROCESS**: declares the pre-processing operations to be performed during build. For example:
			/* Extract the attribute columns of the *$j* table from *$pro* and map the result to *$attr* for subsequent use. */
			PREPROCESS: $attr:(find_attributes $j $pro)
			• **CTE**: matches CTEs.
			• **PROJECT**: matches projection columns.
			• **FROM**: matches the FROM expression, which can be an ordinary table, view, subquery, or join.
			• **FILTER**: matches filter conditions.
			• **GROUP**: matches GROUP BY and HAVING conditions.
			• **SORT**: matches the ORDER BY condition.
			• **LIMIT**: matches the LIMIT and OFFSET conditions.
			• **POSTPROCESS**: declares the post-processing operations to be performed during build. For example:
			/* Delete the useless table information of the current QUERY BLOCK specified by THIS. */
			POSTPROCESS: (remove_unused_table THIS)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description
			• **MAINBLOCK**: specifies whether the query block is in the outermost layer.

JOIN	JOIN<JOIN_TYPE> {
	ALIAS: ($alia: *)
	LEFT: ($left_expr: *)
	RIGHT: ($right_expr: *)
	ON: ($on_expr: *)
	ONLY_TOP: false
	
	RECURSE_MODE($MODE)
	}
		• JOIN_TYPE: The following join types are supported: LEFT JOIN, RIGHT JOIN, INNER JOIN, CROSS JOIN, SEMI JOIN, ANTI JOIN, and FULL JOIN. The options are as follows:
		LEFT (used for both LEFT JOIN and RIGHT JOIN), **INNER**, **CROSS**, **SEMI**, **ANTI**, and **FULL**.
		• This keyword has no default type and must be explicitly specified; otherwise, a parsing error will be reported.
			This keyword matches the JOIN content. The inherent parameters are as follows:
			• **ALIAS**: alias of the JOIN.
			• **LEFT**: left expression in the JOIN.
			• **RIGHT**: right expression in the JOIN. This parameter needs to be omitted when **JOIN_TYPE** is set to **CROSS**.
			• **ON**: JOIN condition. This parameter can be omitted, indicating that the parameter is not matched.
			• **ONLY_TOP**: In the scenario where multiple JOINs are used, only the first JOIN node is matched and recursive matching is not performed. The value can be **true** or **false**. If this parameter is omitted, the default value **false** is used.
			• **RECURSE_MODE**: recursive mode. The value can be **CURRENTFIRST** (pre-order traversal), **CHILDFIRST** (post-order traversal), or **CURRENTONLY** (matching only the top-level node). This parameter can be omitted. If it is omitted, the default value **CURRENTFIRST** is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

GROUP	GROUP<> {
	GROUP: ($group_expr: *)
	FILTER: ($filter_expr: *)
	GROUPSET: ($groupset_expr: *)
	}
		No type information.	This keyword matches a GROUP BY structure. The inherent parameters are as follows:
			• **GROUP**: matches the GROUP BY condition.
			• **FILTER**: matches the HAVING condition. This parameter can be omitted, indicating that the parameter is not matched.
			• **GROUPSET**: matches the GROUPSET condition. This parameter can be omitted, indicating that the parameter is not matched.

LIMIT	LIMIT<> {
	LIMIT: ($limit_expr: *)
	OFFSET: ($offset_expr: *)
	}
		No type information.	This keyword matches a LIMIT structure. The inherent parameters are as follows:
			• **LIMIT**: matches the LIMIT condition. This parameter can be omitted, indicating that the parameter is not matched.
			• **OFFSET**: matches the OFFSET condition. This parameter can be omitted, indicating that the parameter is not matched.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

AGG	AGG<AGG_TYPE> {
	ARGS: ($exprs: *)
	DISTINCT: *
	}
		• AGG_TYPE: All built-in aggregate functions can be matched. Multiple values can be specified at the same time and separated by vertical bars (|).
		• If this keyword type is not explicitly specified, the default type is used, that is, all types of aggregate functions will be matched.
			This keyword can match all aggregate functions. You can specify the type to match a specific aggregate function. For example, **AGG<>** indicates that all aggregate functions are matched, and **AGG<MIN|MAX>** indicates that only the MIN and MAX functions are matched. The inherent parameters are as follows:
			• **AGG_TYPE**: Aggregate function type. Separate multiple types with vertical bars (|). Aggregate functions include those in the PG_PROC system catalog.
			• **ARGS**: input parameters of the aggregate function.
			• **DISTINCT**: specifies whether to add the DISTINCT constraint when an aggregate function is created. The value can be **TRUE** or **FALSE**. The default value is **FALSE**.
			NOTE
			The **ARGS** parameter can be omitted in the AGG keyword matching. When this parameter is omitted, other parameters must also be omitted. The following are two scenarios:
			• Specify **AGG_TYPE** to match the aggregate function corresponding to **AGG_TYPE**.
			/* Match the aggregate functions of the MIN, MAX, and SUM types. */
			AGG<MIN|MAX|SUM> {}
			• Omit **AGG_TYPE** to match all aggregate functions.
			AGG<> {}

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

SUBLINK	SUBLINK<SUBLINK_TYPE> {
	TEST_EXPR: ($test: *)
	QUERY: ($exprs: *)
	}
		• SUBLINK_TYPE: ANYSUBLINK, NOTANYSUBLINK, ALLSUBLINK, NOTALLSUBLINK, EXISTSSUBLINK, NOTEXISTSSUBLINK, EXPRSUBLINK, and ROWCOMPARESUBLINK are supported.
		• This keyword has no default type and must be explicitly specified; otherwise, a parsing error will be reported.
			This keyword matches a SUBLINK structure. The inherent parameters are as follows:
			• **QUERY**: matches a QUERY structure.
			• **TEST_EXPR**: matches the TEST expression. This parameter is valid only for the ANYSUBLINK, NOTANYSUBLINK, ALLSUBLINK, NOTALLSUBLINK, and ROWCOMPARESUBLINK types.

CROSS	CROSS<> {
	ARGS: ($exprs: *)
	}
		No type information.	This keyword matches a Cartesian product. The inherent parameters are as follows:
			ARGS: table information of the Cartesian product.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

SET	SET<SET_TYPE> {
	LEFT: ($left_expr: *)
	RIGHT: ($right_expr: *)
	
	MATCHTYPE_ALLSAME: *
	
	IGNOREUNMATCHTYPE: *
	}
		• *SET_TYPE*: UNION, UNIONALL, INTERSECT, INTERSECTALL, EXCEPT, and EXCEPTALL are supported.
		• When keywords are used for matching, multiple types can be combined. Different types are separated by vertical bars (|), for example, UNION | UNIONALL.
			This keyword matches a set. The inherent parameters are as follows:
			• **LEFT**: left expression of the SET keyword.
			• **RIGHT**: right expression of the SET keyword.
			• **MATCHTYPE_ALLSAME**: specifies whether to check the consistency of all operation types in a set. If this parameter is set to **TRUE**, all operation types in the set must be the same. This parameter is optional. The default value is **TRUE**.
			• **IGNOREUNMATCHTYPE**: specifies whether to ignore the unsupported operation types in a set. If this parameter is set to **TRUE**, the set can contain operation types other than the matched ones. This parameter is optional. The default value is **FALSE**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

			NOTE
			The SET keyword matching supports the general template matching mode. In this mode, the RIGHT keyword and its expression information can be omitted. When there are multiple subqueries, this mode can match all nodes that meet the expression in the LEFT keyword at a time. The fixed format is as follows (**MATCHTYPE_ALLSAME** is set to the default value **TRUE**, and **IGNOREUNMATCHTYPE** is set to the default value **FALSE**):
			`SET<xxx>`
			`{`
			`LEFT: $set_left : QUERY<xxx> {xxx}`
			`}`
			This can be used together with the FOREACH_NODES_WITH_TEMPLATE function. The usage is as follows:
			`FOREACH_NODES_WITH_TEMPLATE $set_left (xxx_func)`

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

PROJECT	PROJECT<> {
	ARGS: ($exprs: *)
	MATCHALL: *
	}
		No type information.	This keyword matches projection columns. The inherent parameters are as follows:
			• **ARGS**: projection column expression.
			• **MATCHALL**: specifies whether to return all matched projection columns. If this parameter is set to **TRUE**, all projection columns that match the projection column expression are returned. If this parameter is set to **FALSE**, only the first matched projection column is returned. This parameter is optional. The default value is **TRUE**.
			NOTE
			The fixed usage of the PROJECT keyword is provided below. The return value is the information about the projection columns that are matched in batches. **MATCHALL** is omitted and uses the default value **TRUE**.
			$proj:PROJECT<>
			{
			ARGS: $exprs: *
			}
			This can be used together with the FOREACH_NODES_WITH_TEMPLATE function. The usage is as follows:
			FOREACH_NODES_WITH_TEMPLATE $proj (xxx_func)

CTE	CTE<$name:*> {
	QUERY: $CTE_QUERY
	}
		No type information.	This keyword matches a CTE structure. The inherent parameters are as follows:
			• **name**: CTE name.
			• **QUERY**: matches a QUERY structure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-21 Extended expression keywords

Keyword	Syntax	Type	Description

OR	OR<> {
	ARGS: ($exprs: *)
	}
		No type information.	This keyword matches an OR expression structure. The inherent parameters are as follows:
			ARGS: content of the OR expression.

AND	AND<> {
	ARGS: ($exprs: *)
	}
		No type information.	This keyword matches an AND expression structure. The inherent parameters are as follows:
			ARGS: content of the AND expression.

NOT	NOT<> {
	ARGS: ($exprs: *)
	}
		No type information.	This keyword matches a NOT expression structure. The inherent parameters are as follows:
			ARGS: content of the NOT expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

BOOLEAN	BOOLEAN<BOOLEAN_TYPE> {
	ARGS: ($exprs: *)
	IGNORENULL: *
	AUTOMATCH: *
	}
		• BOOLEAN_TYPE:
		The value can be **AND**, **OR**, **NOT**, or any combination of them separated by vertical bars (|).
		• If the keyword type is not explicitly specified, the default type is used, which matches all BOOL types.
			This keyword matches a BOOL expression structure. The inherent parameters are as follows:
			• **ARGS**: content of a BOOL expression.
			• **IGNORENULL**: specifies whether an empty expression is considered valid. The value can be **TRUE** (valid) or **FALSE** (invalid). The default value is **FALSE**, indicating that the parameter can be omitted. If it is omitted, the default value is used.
			• **AUTOMATCH**: specifies whether to automatically match BOOL expressions from the given expressions. The value **TRUE** indicates that BOOL expressions contained in the given conditions are automatically matched. The value **FALSE** indicates that the given expressions must be complete BOOL expressions. The default value is **FALSE**, indicating that the parameter can be omitted. If it is omitted, the default value is used.

CONST	CONST<CONST_TYPE> {
	ARGS: ($exprs: *)
	}
		• CONST_TYPE: The INTEGER and BOOL types are supported.
		• If this keyword type is not explicitly specified, only the INTEGER type will be matched.
			This keyword matches the content of a CONST expression. The inherent parameters are as follows:
			ARGS: content of the CONST expression, which can be left empty.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

NULLTEST	NULLTEST<NULLTEST_TYPE> {
	ARGS: ($exprs: *)
	}
		• NULLTEST_TYPE:
		The ISNULL and NOTNULL types are supported.
		• This keyword has no default type and must be explicitly specified; otherwise, a parsing error will be reported.
			This keyword matches an IS NULL or NOT NULL expression. The inherent parameters are as follows:
			ARGS: content of the NULLTEST expression.

OPERATOR	OPERATOR<OPERATOR_TYPE>{
	LEFT: ($left_expr: *)
	RIGHT: ($right_expr: *)
	IGNORESIDE: true
	}
		• OPERATOR_TYPE:
		The EQ, NE, GT, GE, LT, LE, and PLUS types are supported. A combination of multiple types is supported. Use vertical bars (|) to separate different types.
		• This keyword has no default type and must be explicitly specified; otherwise, a parsing error will be reported.
			This keyword matches an OPERATOR expression. The inherent parameters are as follows:
			• **LEFT**: left expression of the OPERATOR expression.
			• **RIGHT**: right expression of the OPERATOR expression.
			• **IGNORESIDE**: specifies whether to ignore the order of the left and right nodes. This parameter can be omitted. If it is omitted, the default value **TRUE** is used.

CASE	CASE<>{
	ARGS: ($expr: *)
	DEFRESULT: ($def: *)
	}
		No type information.	This keyword matches a CASE expression. The inherent parameters are as follows:
			• **ARGS**: matched expression.
			• **DEFRESULT**: default result expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Type	Description

WHEN	WHEN<>{
	ARGS: ($expr: *)
	DEFRESULT: ($def: *)
	}
		No type information.	This keyword matches a WHEN expression. The inherent parameters are as follows:
			• **ARGS**: matched WHEN condition.
			• **DEFRESULT**: expression of the returned result.

COL	COL<> {
	ARGS: ($exprs: *)
	}
		No type information.	This keyword matches the attribute column expression of a table. The inherent parameters are as follows:
			ARGS: content of the COL expression.

Table 7-22 Additional keywords

Keyword	Syntax	Description

CONSTRAINTS	CONSTRAINTS: /*
	CONTENT */
		This keyword is used to restrict the structure keyword or expression extension keyword, and can be used as the last input parameter of the keywords in **Table 7-20** and **Table 7-21**. For example:
		/* Restrict the QUERY keyword to contain the LIMIT clause. */
		QUERY<> {
		PROJECT: $proj:*
		FILTER: $sub_filter:*
		CONSTRAINTS: (check_has_constraint THIS LIMIT)
		}

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Syntax	Description
TEMPLATE	TEMPLATE { /* CONTENT */ }	This keyword is used to restrict the input parameters of a function or an expression calculation function. The content cannot be empty. It can be a structure keyword or expression extension keyword. When the function implementation needs to use a template instead of the constructed expression, this keyword can be used. For details about the keywords supported by each function, see the description of each function. For example: /* **find_nodes_with_template** uses the template {DSLSTRUCT} to match the related content in *$redundant_info* and maps the result to *$r_info_template* for subsequent use. If the TEMPLATE keyword is not used in DSLSTRUCT, the template is parsed as an expression instead of a template. */ $r_info_template: (find_nodes_with_template $redundant_info template {DSLSTRUCT})
THIS	THIS	This keyword is used to restrict the input parameters of a function or an expression calculation function and refers to the query block of the current scope. For example: /* Delete the useless table information of the current QUERY BLOCK specified by THIS. */ POSTPROCESS: (remove_unused_table THIS)
ROOT	ROOT	This keyword is used to restrict the input parameters of the function or expression calculation function and refers to the query block at the first layer. For example: /* Restrict ROOT to refer to the first-layer QUERY BLOCK that contains the LIMIT clause. */ CONSTRAINTS: (check_has_constraint ROOT LIMIT)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-23 Constraint functions

Function	Prototype	Parameter	Description
CHECK_HAS_CONSTRAINT	bool check_has_constraint(DSLNode $args, DSLType $flags)	• *$args*: Type: DSLNode Description: specifies the DSL node to be checked (corresponding to the query tree or a node). • *$flags*: Description: specifies the check type. Only the existence of the corresponding constraints is checked. The following constraints are supported: VOLATILE (VOLATILE function), STABLE (STABLE function), IMMUTE (IMMUTABLE function), AGG (aggregate function), WINAGG (window function), DISTINCT (DISTINCT keyword), DISTINCT_ON (DISTINCT ON keyword), ROWNUM (ROWNUM keyword), MODIFYCTE (CTE table of the UPDATE type), LIMIT (LIMIT constraint), OFFSET (LIMIT	Checks whether the node contains the check types specified by *$flags* and determines whether the constraint conditions are met based on the returned result.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		OFFSET constraint), SET (set type), GROUP (GROUP BY condition), GROUPSET (GROUPING SETS condition), HAVING (HAVING condition), ROWMARK (scenarios with row locks such as FOR UPDATE), CTE (CTE list), RETURNSET (functions that return multiple columns), SUBLINK (sublinks), PLACEHOLDERVAR (placeholder variables), UNPIVOT (row-column conversion), UPPERVAR (upper-layer parameters, which are generally used to determine whether related sublinks exist), SORT (ORDER BY), RECURSIVE_CTE (recursive CTEs), and FULLGROUP (the table attribute column in the projection columns must be in the group).	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
CHECK_IS_SINGLE_TABLE	bool check_is_single_table(DSLNode $args)	*$args*: Type: DSLNode Description: specifies the DSL node to be checked.	Checks whether the node is a single table.

Function	Prototype	Parameter	Description
CHECK_IS_EQUAL	bool check_is_equal(DSLNode $args1, DSLNode $args2, DSLType $flags)	• *$args1*: Type: DSLNode Description: specifies the DSL node 1 to be checked for equality. • *$args2*: Type: DSLNode Description: specifies the DSL node 2 to be checked for equality. • *$flags*: Type: DSLType Description: specifies whether a strict check is performed. The value can be **STRICT** or **PHYSIC**. If it is omitted, the default value **STRICT** is used. – If the value is **STRICT**, all member variables of the nodes to be checked must be equal. – If the value is **PHYSIC**, variables are regarded as equal as long as they come from the same type of tables, the tables are located in the same position, and the parameter types are the same.	Checks whether nodes 1 and 2 meet the equality condition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
CHECK_IS_SUBSET	bool check_is_subset(List $sub, List $parent)	• *$sub*: Type: List Description: specifies the set to be checked. • *$parent*: Type: List Description: specifies the parent set for check.	Checks whether a set is a subset of another set.
CONTAIN_ATTR_OF	bool contain_attr_of(List $args, List $tables)	• *$args*: Type: List Description: specifies the set of attribute columns to be checked. • *$tables*: Type: List Description: specifies a table node, which can be obtained by calling FIND_TABLES.	Checks whether the attribute columns of the specified table are contained.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
CHECK_TYPE_FOREACH	bool check_type_foreach(List $arg, DSLType $check_type, DSLType $any_all_type)	• *$arg*: Type: List or DslNode Description: specifies the set of expressions to be checked. • *$check_type*: Type: DSLType Description: specifies the expressions supported, including AND (AND expression), OR (OR expression), NOT (NOT expression), CONST (constant type), COUNT_STAR (COUNT(*) expression), SUBQUERY (subquery), NOTNULL (NOT NULL attribute), and COL (attribute column). • *$any_all_type*: Type: DSLType Description: Select **ALL** or **ANY**. This parameter can be omitted. If it is omitted, the default value **ALL** is used. – **ALL**: All items to be checked must be expressions. – **ANY**: Any check item is an expression.	Checks the expression type.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
CHECK_IS_EMPTY	bool check_is_empty(DSLNode $arg)	*$arg*: Type: DSLNode Description: specifies the expression to be checked.	Checks whether the expression is empty.

Function	Prototype	Parameter	Description
CHECK_CONSTRAINT_FOREACH	bool check_constraint_foreach(List $args, DSLType $flags)	• *$args*: Type: List Description: specifies the table set to be checked, which can be obtained by calling FIND_TABLES. • *$flags*: Description: specifies the check type. Only the existence of the corresponding constraints is checked. The following constraints are supported: WINAGG (window function), DISTINCT (DISTINCT keyword), DISTINCT_ON (DISTINCT ON keyword), MODIFYCTE (CTE table of the UPDATE type), LIMIT (LIMIT constraint), OFFSET (LIMIT OFFSET constraint), SET (set type), GROUP (GROUP BY condition), GROUPSET (GROUPING SETS condition), HAVING (HAVING condition), ROWMARK (scenarios with row locks such as	Checks whether all tables contain the constraints specified by *$flags*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		FOR UPDATE), UNPIVOT (row-column conversion), and SORT (ORDER BY).	
CHECK_IS_SAME_TABLE	bool check_is_same_table(List $args1, List $args2)	• *$args1*: Type: List Description: specifies the table set 1 to be checked, which can be obtained by calling FIND_TABLES. • *$args2*: Type: List Description: specifies the table set 2 to be checked, which can be obtained by calling FIND_TABLES.	Checks whether table sets are equal.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
CHECK_IS_SIMPLE_COLUMN	bool check_is_simple_column(List $args1, List $dsl_simple_col_type_list, DSLTpye $any_all_type)	• *$args1*: Type: List Description: specifies the set of nodes to be checked. • *$dsl_simple_col_type_list:* Type: List Description: specifies the type of the node that can pass the check. Options: **COL**, **CONST**, **DSLEXPR**, **IGNOREAGG**, or any of their combinations separated by vertical bars (|). • *$any_all_type*: Type: DSLType Description: specifies the behavior when a structure that meets the conditions is detected. Options: **ALL** and **ANY**. – **ALL**: Returns **true** when all items to be checked meet the conditions. – **ANY**: Returns **true** when one item to be checked meets the conditions.	Checks whether the node in the list is of the specified node type. • **COL** (a single attribute column of a table) • **CONST** (a single constant value) • **EXPR** (a single simple expression, for example, t.a = 1) • **IGNOREAGG** (ignore the AGG expression)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-24 Expression calculation functions

Function	Prototype	Parameter	Description
EXPR_COMPUTE	Node expr_compute(Node $args)	• Input parameter: *$args*: Type: Node Description: specifies the expression in the query tree. • Output parameter: *$result*: Type: Node Description: specifies the expression after constant calculation.	Calculates constant expressions, for example, preprocessing the expressions that can be calculated as constants to facilitate subsequent logic analysis. Example: Calculate "1 + 1" in the expression in advance to obtain the constant value **2**, or calculate "1 = 0" in advance to obtain the constant value **false**.
BUILD_PROJECT	Node build_project(Node $args)	• Input parameter: *$args*: Type: Node Description: specifies the table attribute columns, which can be obtained by calling functions such as FIND_ATTRIBUTES. • Output parameter: *$result*: Type: Node Description: specifies the constructed projection column set.	Constructs a projection column set based on the table attribute columns passed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_TABLE	Node build_table(Node $args)	• Input parameter: *$args*: Type: QUERY or Node Description: specifies the QUERY structure or return value of **FIND_TABLES**. • Output parameter: *$result*: Type: Node Description: specifies the newly constructed table.	Constructs a new table based on the input parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_BOOL_EXPR	Node build_bool_expr(DSLType $bool_type, Node $args1, DSLNode $args2)	• Input parameters: – *$bool_type*: Type: DSLType Description: specifies AND or OR to construct an AND expression or an OR expression. – *$args1*: Type: Node Description: specifies parameter 1 of the expression. Generally, it is a linked list. This function uses the elements in the linked list to construct a Boolean expression. – *$args2*: Type: DSLNode Description: specifies parameter 2 of the expression, that is, the template used to construct the expression. This function supports the following types of TEMPLATE node: NULLTEST. If this parameter is left empty, the elements in **$args1** are directly used to	Constructs an expression of a specified type.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		construct a Boolean expression. For example: /* Use the elements in $v_list to construct an AND expression. */ (build_bool_expr AND $v_list) If this parameter is not empty, the elements in **$args1** are used to construct an expression based on the TEMPLATE specified by **$args2**, and then this expression is used to construct the final Boolean expression. For example: /* Use the elements in $v_list to construct an ISNULL expression, and then use the constructed expression to construct an AND expression. */ (build_bool_expr AND $v_list TEMPLATE {NULLTEST<ISNULL>{args: ELEMENT}}) • Output parameter: *$result*: Type: Node Description: specifies the constructed expression.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_GROUP	Node build_group(Node $args)	• Input parameter: *$args*: Type: Node Description: specifies the table attribute columns, which can be obtained by calling functions such as FIND_ATTRIBUTES. • Output parameter: *$result*: Type: Node Description: specifies the constructed grouping set.	Constructs grouping information based on the passed table attribute columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_FROMEXPR_BY_QUERY	Node build_fromexpr_by_query(Query $query, Node $table)	• Input parameters: – *$query*: Type: Query Description: specifies the QUERY structure to which the constructed table definition belongs. The structure contains table construction information. – *$table*: Type: Node Description: specifies the table to be created, which is usually a part of the original table definition and is obtained through matching. • Output parameter: *$result*: Type: Node Description: specifies the expression of the constructed target table definition.	Creates the FROM expression of the QUERY structure based on the passed table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REMOVE_DISTINCT	bool remove_distinct(DSLExpr $query)	• Input parameter: *$args*: Type: DSLExpr or Query Description: The value can be **THIS**, **ROOT**, or **QUERY**. – **THIS**: query at the current layer. – **ROOT**: root query. – **QUERY**: target query. • Output parameter: *$result*: Type: bool Description: **TRUE** indicates that the removal is successful. **FALSE** indicates that the removal fails.	Removes the DISTINCT part of the target query.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REMOVE_UNUSED_TABLE	bool remove_unused_table(DSLExpr $query)	• Input parameter: *$args*: Type: DSLExpr or Query Description: The value can be **THIS**, **ROOT**, or **QUERY**. – **THIS**: query at the current layer. – **ROOT**: root query. – **QUERY**: target query. • Output parameter: *$result*: Type: bool Description: **TRUE** indicates that the removal is successful. **FALSE** indicates that the removal fails.	Removes irrelevant tables from the target query. This function needs to be executed when the FROM part of the query is modified or rebuilt.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_UNIQUE_ATTR	List find_unique_attr(List $args, UniqueType $type)	• Input parameters: – *$args*: Type: List Description: specifies a set of table attribute columns. – *$type*: Type: UniqueType Description: specifies the unique key type. The value can be **NotNullUnique** (NOT NULL unique index) or **Unique** (unique index). • Output parameter: *$result*: Type: List Description: Returns the attribute value that meets the specified conditions. If there are multiple attribute values (for example, unique indexes), only the first matched constraint is returned.	Checks whether the base table of the specified table contains a unique key.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_TABLES	List find_tables(DSLNode $args, TableType $type, DSLType $include_join)	• Input parameter: – *$args*: Type: DSLNode Description: specifies the information about the original node to be extracted. – *$type*: Type: TableType Description: specifies the type of the table to be extracted. – **ALL**: all table types – **BASEREL**: base table – **QUERY**: subquery – **JOIN**: table of the JOIN node – **FUNCTION**: function table – **VALUES**: VALUES table – **CTE**: CTE table – **TBLRESULT**: RESULT table – **XMLTABLE**: XML table – *$include_join*: Type: DSLType Description: specifies whether to include the	Extracts all tables from the node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		upper-layer JOIN node. **INCLUDEJOIN**: yes. **NOTINCLUDEJOIN**: no. This parameter can be omitted. If it is omitted, the default value **NOTINCLUDEJOIN** is used. • Output parameter: *$result*: Type: List Description: Returns table information.	

Function	Prototype	Parameter	Description
FIND_ATTRIBUTES	List find_attributes(DSLNode $tablelist, DSLNode $args) List find_attributes(DSLNode $tablelist, DSLNode $args, DSLNode $table_query, DSLNode $find_attri_marks, DSLNode $do_dedup)	• Input parameters: – *$tablelist*: Type: DSLNode Description: specifies the set of target tables to be extracted. It can be obtained using the find_tables function. – *$args*: Type: DSLNode Description: specifies the information about the original node to be extracted. – *$table_query*: Type: DSLNode Description: specifies the query to which the table set belongs. Options: **THIS** (query at the current layer), **ROOT** (main query), or a variable (starting with *$* and obtained in the matching phase). – *$find_attri_marks*: Type: DSLNode Description: specifies the attribute restrictions that the extracted column should meet.	Extracts related table attributes from the node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		Default value: **ALL**. Options: **ALL** (no attribute requirement), **NOTNULL** (NOT NULL attribute), **UNIQUE** (UNIQUE attribute), and any of their combinations. – *$do_dedup*: Type: DSLNode Description: specifies whether to deduplicate the extracted results. Default value: **DEDUPLICATE**. Options: **DEDUPLICATE** (deduplication) and **NOTDEDUPLICATE** (no deduplication). ● Output parameter: *$result*: Type: List Description: Returns table information.	

Function	Prototype	Parameter	Description
APPEND	List append(List $arg1, List $arg2)	• Input parameter: – *$arg1*: Type: List or Node Description: specifies a node list. – *$arg2*: Type: List or Node Description: specifies a list or single node. • Output parameter: Type: List Description: specifies a list or single node.	Adds another node or list to a list, or constructs a list from two nodes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REPLACE_WITH	Node replace_with(DSLNode $arg1, DSLNode $arg2, DSLNode $arg3) bool replace_with(DSLNode $arg1, DSLNode $arg2, DSLNode $arg3, DSLNode $replace_node_option) bool replace_with(DSLNode $arg1, DSLNode $arg2, DSLNode $arg3, DSLNode $replace_node_option, DSLNode $replace_node_mode)	• Input parameter: – *$arg1*: Type: DSLNode Description: specifies the parent node of the node to be replaced. – *$arg2*: Type: DSLNode Description: specifies the node to be replaced. – *$arg3*: Type: DSLNode Description: specifies the node whose content is to be replaced. – *$replace_node_option*: Type: DSLNode Description: specifies the replacement mode when the new node replaces the old node. Default value: **ASNEW**. Options: **DIRECT** (shallow copy) and **ASNEW** (deep copy). This parameter can be omitted. If it is omitted, the default value **ASNEW** is used. – *$replace_node_mode*:	Replaces the *$arg2* node under the parent node *$arg1* with the *$arg3* node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		Type: DSLNode Description: specifies the behavior of traversing the structure to be replaced. For details, see the description of the optional values. Default value: **ANY**. Options: **ANY** (replacing only the first node that meets the condition) and **ALL** (replacing all nodes that meet the condition). This parameter can be omitted. If it is omitted, the default value **ANY** is used. • Output parameter: Type: Node Description: Node after replacement.	

Function	Prototype	Parameter	Description
REPLACE_WITH_REC	Node replace_with_rec (DSLNode $arg1, DSLNode $arg2, DSLNode $arg3) bool replace_with_rec (DSLNode $arg1, DSLNode $arg2, DSLNode $arg3, DSLNode $replace_node_option)	• Input parameters: – *$arg1*: Type: DSLNode Description: specifies the parent node of the node to be replaced. – *$arg2*: Type: DSLNode Description: specifies the node to be replaced. – *$arg3*: Type: DSLNode Description: specifies the node whose content is to be replaced. – *$replace_node_option*: Type: DSLNode Description: specifies the replacement mode when the new node replaces the old node. Default value: **ASNEW**. Options: **DIRECT** (shallow copy), **ASNEW (deep copy), and UPDATE_COL_REF** (updating of the **varno** attribute in the Var structure). This parameter can be omitted. If it is omitted,	Replaces the *$arg1* node found in the parent node recursively with the *$arg3* node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		the default value **ASNEW** is used. • Output parameter: Type: Node Description: Node after replacement.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REPLACE_TABLE	Node replace_table (QUERY $arg1, DSLNode $arg2, DSLNode $arg3, DSLNode $arg4 = DIRECT)	• Input parameters: – *$arg1*: Type: QUERY Description: specifies the QUERY node of the node to be replaced. – *$arg2*: Type: DSLNode Description: specifies the table node to be replaced. – *$arg3*: Type: DSLNode Description: specifies the table node used for replacement. – *$arg4*: Type: DSLNode Description: specifies the table replacement mode. This parameter is optional. If this parameter is not explicitly specified, the default value **DIRECT** is used. If this parameter is explicitly specified, the value can only be **ADD**. Default value: **DIRECT**, indicating that the direct replacement mode is used.	Replaces the *$arg2* table node in the *$arg1* QUERY node with the *$arg3* table node according to the mode specified by *$arg4*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		Option: **ADD**, indicating that a new node is added. • Output parameter: Type: Node Description: Node after replacement.	
REPLACE_TABLE_FOREACH	Node replace_table (QUERY $arg1, DSLNode $arg2, DSLNode $arg3)	• Input parameters: – *$arg1*: Type: QUERY Description: specifies the QUERY node of the node to be replaced. – *$arg2*: Type: List Description: specifies the list of table nodes to be replaced. – *$arg3*: Type: DSLNode Description: specifies the table node used for replacement. • Output parameter: Type: Node Description: Node after replacement.	Replaces all table nodes in the *$arg2* collection of the QUERY node with the *$arg3* table node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_EXPR_WITHOUT	List find_expr_without(Expr $args1, Expr $args2)	• Input parameters: – *$args1*: Type: Expr Description: specifies the parent expression node to be processed. – *$args2*: Type: Expr Description: specifies the expression node to be excluded. • Output parameter: Type: List Description: specifies the list of expression nodes remaining after the target expression is excluded.	Searches for the part that does not contain the specified *$args2* expression node from the *$args1* parent expression node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_EXPR_WITH	List find_expr_with(Expr $args, DSLType $flags, DSLType $any_all_type)	• Input parameters: – *$args*: Type: Expr Description: specifies the expression node to be processed. – *$flags*: Type: DSLType Description: specifies the supported types. For instance, **UPPERVAR** indicates that variables of the upper-layer query block are included. – *$any_all_type*: Type: DSLType Description: Extracts expressions that meet the specified conditions. The options are **ALL** and **ANY**. **ALL**: extracts all expressions that meet the specified conditions. **ANY**: extracts the first expression that meets the specified conditions. • Output parameter: Type: List Description: specifies the nodes that meet the	Searches for the part that contains the specified conditions from the *$args* expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		specified conditions.	
LIST_FIRST	Node list_first(List $args)	• Input parameter: *$args*: Type: List Description: specifies the list to be processed. • Output parameter: Type: Node Description: specifies the extracted node.	Extracts the first node from the *$args* list.
LIST_LENGTH	Integer list_length(List $args)	• Input parameter: *$args*: Type: List Description: specifies the list to be processed. • Output parameter: Type: INTEGER Description: specifies the list length.	Obtains the list length.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
PULL_RELA_ATTR	List pull_rela_attr(Expr $args, DSLType $flags)	• Input parameters: – *$args*: Type: Expr Description: specifies the expression node to be processed. – *$flags*: Type: DSLType Description: Extracts expressions at the specified layer. The options are **THIS**, **PARENT**, and **PARENTS**. **THIS**: extracts expressions at the current layer. **PARENT**: extracts expressions at the upper layer. **PARENTS**: extracts expressions at the upper two layers. • Output parameter: Type: List Description: specifies the list of expressions that meet the specified conditions.	Extracts expression nodes that meet the layer conditions in the expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
PULL_EXPR_TO_QUERY	Node pull_expr_to_query(List $args, Query $query)	• Input parameters: – *$arg*: Type: Node Description: specifies the table attribute node to be processed. – *$query*: Type: Query Description: specifies the target query during the pullup. • Output parameter: Type: Node Description: specifies the node after the pullup.	Pulls up the *$arg* table attribute node to the *$query* node.
BUILD_EXPR_THROUGH_PULLUP	Node build_expr_through_pullup(Node $args, Query $query)	• Input parameters: – $args Type: Node Description: specifies the expression node to be processed. – *$query*: Type: Query Description: specifies the target query. The value can be the result of calling build_table, that is, a query is created. • Output parameter: Type: Node Description: specifies the node after the pullup.	Constructs a new expression using the related expression and the new table. The table attributes in the expression are mapped to the new table attributes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REMOVE_QUERY_OPTION	bool remove_query_option(Query $query, DSLType $remove_query_options)	• Input parameters: – *$query*: Type: Query Description: specifies the QUERY structure that needs to be removed. Options: **THIS**, **ROOT**, and **QUERY**. – **THIS**: query at the current layer. – **ROOT**: root query. – **QUERY**: target query. – *$remove_query_options*: Type: DSLType Description: specifies the part to be removed from the QUERY structure. Options: **DISTINCT**, **DISTINCT_ON**, **GROUP**, **SORT**, **LIMIT**, **OFFSET**, and any of their combinations. • Output parameter: Type: bool Description: specifies whether the removal is successful.	Removes the specified part from the QUERY structure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_WHEN_EXPR	CaseWhen build_when_expr(CaseWhen $casewhen)	• Input parameter: *$casewhen*: Type: CaseWhen Description: specifies the CaseWhen structure template to be generated. • Output parameter: Type: CaseWhen Description: specifies the generated CaseWhen structure.	Generates the CaseWhen structure.
BUILD_CASE_EXPR	CaseExpr build_case_expr(CaseExpr $caseexpr)	• Input parameter: *$caseexpr*: Type: CaseExpr Description: specifies the CaseExpr structure template to be generated. • Output parameter: Type: CaseExpr Description: specifies the generated CaseExpr structure.	Generates the CaseExpr structure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_SAME_TABLES	List find_same_tables(List $args, DSLType $type)	• Input parameter: – *$args*: Type: List Description: specifies the list of tables to be processed. – *$type*: Type: DSLType Description: specifies the type of the table to be processed. Option: **BASEREL** (base table) – *$table_check_options*: Type: DSLType Description: specifies the function processing mode in special scenarios. Options: null (default scenario, processed in default mode) and **SUPPORT_SPECIFIED_PARTITION** (specified partition). • Output parameter: Type: List Description: specifies the linked list of OIDs of the same table.	Searches for the same table. By default, partitions cannot be specified. **NOTE** Example of specifying a partition: SELECT * FROM t1 partition(p1), t2 partition(p2) WHERE t1.c1=t2.c1; In this example, the partition keyword is used to specify a partition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_NODES_WITH_TEMPLATE	List find_nodes_with_template(Node $node, DSLNode $template, DSLType $rec_mode)	• Input parameter: – *$node*: Type: Node Description: specifies the structure to be traversed. – *$template*: Type: DSLNode Description: specifies the target template to be matched. The TEMPLATE keyword is used to describe the target template in the format "TEMPLATE *{FUNC_TEMPLATE_NODE}*". – *$rec_mode*: Type: DSLType Description: specifies the traversal processing mode. The value can be **CURRENTFIRST** (pre-order traversal), **CHILDFIRST** (post-order traversal), or **CURRENTONLY** (only matching the top-level node). This parameter can be omitted, indicating the non-recursive mode. When this parameter is not empty,	Traverses the structure to find the structure that matches the target template.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		the input parameter *$template* can only be a JOIN template. For example: /* Use the JOIN template to perform matching in *$fr* in post-order traversal mode. */ find_nodes_with_template $fr template {JOIN<INNER>{ LEFT: $l : * RIGHT: $r : * on: $o : *}} CHILDFIRST • Output parameter: Type: List Description: specifies the linked list of structures that match the target template.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FOREACH_NODES_WITH_TEMPLATE	bool foreach_nodes_with_template(DSLNode $arg1, DSLNode $arg2)	• Input parameter: – *$arg1*: Type: DSLNode, specifically DSLExpr. The result returned by the find_nodes_with_template function is stored in the linked list. Description: specifies the set of structures to be processed. – *$arg2*: Type: DSLNode Description: specifies the method of processing structures. • Output parameter: Type: bool Description: specifies whether the function is executed successfully.	Traverses each element in the linked list and processes each element in the specified mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
BUILD_SUBLINK	Sublink build_sublink(Node $sublink) Sublink build_sublink(Node $sublink, List $columns, List $operator)	• Input parameters: – *$sublink*: Type: Sublink or Sublink nested with a BoolExpr of the NOT type Description: specifies the SubLink structure template to be built. – *$columns*: Type: List Description: specifies the columns in the ANY, ALL, or ROWCOMPARE sublink to be compared with those of the sublink. – *$operator*: Type: List Description: specifies the operation to be performed in the ANY, ALL, and ROWCOMPARE sublinks. Options: **EQ**, **NE**, **GT**, **GE**, **LT**, **LE**, **PLUS**, and **ANY**. • Output parameter: – Type: Sublink – Description: specifies the constructed SubLink structure.	Constructs a SubLink structure. • For a sublink of the ANY, ALL, or ROWCOMPARE type, use a function with three input parameters. • For a sublink of any other type, use a function with one input parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
DSLNODE_COPY	Node dslnode_copy(Node $node, DSLType $copy_node_option)	• Input parameters: – *$node*: Type: Node Description: specifies the DSLNode structure to be copied. – *$copy_node_option*: Type: DSLType Description: specifies the copy mode. Options: **DIRECT** (shallow copy) and **ASNEW** (deep copy). This parameter can be omitted. If it is omitted, the default value **DIRECT** is used. • Output parameter: Type: Node Description: specifies the DSLNode structure generated after copying.	Copies the DSLNode structure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_EQUALCLASS_VARS	List find_equalclass_vars(BoolExpr $exprs, DSLNode $bool_expr_type, DSLNode $operator_type)	• Input parameters: – *$exprs*: Type: set of OpExprs. Currently, BoolExpr of the AND type is supported. Description: specifies the expression based on which the equivalence class is generated. – *$bool_expr_type*: Type: DSLNode Description: specifies the logical relationship types between multiple expressions supported during equivalence class generation. Option: **AND** – *$operator_type*: Type: DSLNode Description: specifies the OpExpr type supported during equivalence class generation. Option: **EQ** • Output parameter: Type: two-dimensional list. Each sublist represents an	Generates equivalence classes based on expressions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		equivalence class. The elements in the sublist are Vars, which have equivalence class relationships. Description: specifies the linked list of equivalence classes.	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
FIND_REDUNDANT_INFO	List find_redundant_info(List $equalclass_list)	• Input parameter: *$equalclass_list*: Type: two-dimensional list. Each sublist represents an equivalence class. The elements in the sublist are Vars, which have equivalence class relationships. Description: specifies the equivalence class information used for generating redundant information. The information is generated by the find_equalclass_vars function. • Output parameter: Type: List. The internal element is the RedundantInfo structure, which contains information such as required tables and redundant tables. Description: specifies the redundant information generated based on equivalence class information, which is used to eliminate redundant tables in some optimization rules.	Generates redundant information based on equivalence class information. This helps eliminate redundant tables in some optimization rules.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
REMOVE_USELESS_CONDITION	BoolExpr remove_useless_condition(Query $query, BoolExpr $expr, DSLType $scenario_option, List $vars)	• Input parameter: – *$query*: Type: Query Description: specifies the QUERY structure that needs to be removed. Options: **THIS**, **ROOT**, and **QUERY**. – **THIS**: query at the current layer. – **ROOT**: root query. – **QUERY**: target query. – *$expr*: Type: BoolExpr Description: specifies the set of predicate conditions to be processed. – *$scenario_option*: Type: DSL_TYPE Description: specifies the scenario to be processed. Option: **SAME_VAR_EQUAL_OPEXPR** (same variable compared for equality, for example, var1=var1) – *$vars*: Type: List Description: specifies the list	Deletes redundant predicate conditions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Prototype	Parameter	Description
		of processed variables. • Output parameter: Type: BoolExpr Description: specifies the BoolExpr after redundant predicate conditions are removed.	

7.15.2.2 DSL Rule Examples and Description

NOTE

This section provides examples to illustrate DSL rule matching. The actual execution result may vary.

Scenario

- Description: When two expression sublinks containing the Count aggregate function are compared for equality, if one sublink can be expressed as a substructure of the other sublink and the filter conditions differ only in an EXISTS sublink condition, optimization can be performed. In principle, whether the values of two COUNT functions equal depends on the **TRUE** result of other conditions only when **FALSE** is returned for the EXISTS condition. When other conditions are **TRUE**, the filter conditions of the two sublinks are opposite according to the semantics of the AND condition, and the COUNT() values must be different. Otherwise, the values are the same. Therefore, in this scenario, NOT EXISTS can be used for equivalent rewriting to reduce the number of sublinks.
- Example SQL statements:

  Before rewriting:

```
SELECT t1.a,
   t1.b
FROM table1 as t1,
   table2 as t2
WHERE t1.a = t2.a
   AND t1.b = t1.b
   AND (
      SELECT COUNT(*)
      FROM table1 as t11
      WHERE t11.a = t1.a
         AND t11.b = t1.b
   ) = (
      SELECT COUNT(*)
      FROM table1 as t21
      WHERE t21.a = t1.a
         AND t21.b = t1.b
         AND EXISTS(
            SELECT 1
            FROM t3
            WHERE t1.a = t3.a
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        )
  );
```

After rewriting:

```
SELECT t1.a,
  t1.b
FROM table1 as t1,
  table2 as t2
WHERE t1.a = t2.a
  AND t1.b = t1.b
  AND NOT EXISTS (
    SELECT 1
    FROM table1 as t11
    WHERE t11.a = t1.a
      AND t11.b = t1.b
      AND NOT EXISTS(
        SELECT 1
        FROM t3
        WHERE t1.a = t3.a
      )
  );
```

DSL Rule

```
QUERY <> {
  FILTER: $exprs: * &&
          (AND<>{
            ARGS: $and_exp: * && ($eq_expr: OPERATOR<EQ>{
              LEFT: SUBLINK<EXPRSUBLINK>{
                QUERY: $ori_query: QUERY<>{
                PROJECT: $proj1: AGG<COUNT>{ARGS: $agg_args1: *}
                FROM: $input1: * && ($sub_l_t: (find_tables $input1 ALL))
                FILTER: $FILTER1 : * && (AND<> {ARGS: $f_1 : * && (OPERATOR<EQ>{
                  LEFT:$f_1_l : *
                  RIGHT:$f_1_r : *
                  CONSTRAINTS: ((check_has_constraint $f_1_l UPPERVAR)
                    && !(check_has_constraint $f_1_r UPPERVAR)) ||
                    (!(check_has_constraint $f_1_l UPPERVAR)
                    && (check_has_constraint $f_1_r UPPERVAR)) &&
                    !(check_has_constraint THIS MODIFYCTE | VOLATILE)
                  })})
                GROUP: GROUP<> {
                  GROUP: $gr1: * && ($att_list: (find_attributes $gr1 THIS))
                  FILTER: $h_c1: *
                  GROUPSET: $gr_s1: *
                  }
                SORT: $s1 : *
                    }
                    }
              RIGHT: SUBLINK<EXPRSUBLINK>{
                QUERY: QUERY<>{
                  PROJECT: $proj2 : AGG<COUNT>{ARGS: $agg_args2: *}
                  FROM: $input2: * && ($sub_r_t: (find_tables $input2 ALL))
                  FILTER: $FILTER2 : * && (AND<> {ARGS: $f_2 : *
                        && ($orther_f: SUBLINK<existssublink>{QUERY: $ssub_query: *})})
                    && ($same_set : (find_expr_without $f_2 $orther_f))
                  GROUP: GROUP<> {
                    GROUP: $gr2: *
                    FILTER: $h_c2: *
                    GROUPSET: $gr_s2: *
                  }
                  SORT: $s2 : *
                  CONSTRAINTS: (check_is_equal $same_set $f_1)
                    && (check_is_equal $gr1 $gr2)
                    && (check_is_equal $h_c1 $h_c2) && (check_is_equal $gr_s1 $gr_s2)
                    && (check_is_equal $s1 $s2)
                    && (((check_type_foreach $proj1 COUNT_STAR)
                    && (check_type_foreach $proj2 COUNT_STAR)) ||
                    ((check_type_foreach (expr_compute $agg_args1) CONST)
                    && (check_type_foreach (expr_compute $agg_args2) CONST)) ||
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                                ((check_type_foreach $agg_args1 COL)
                                && (check_type_foreach $agg_args2 COL)
                                && (check_is_equal $agg_args1 $agg_args2)))
                                && !(check_has_constraint THIS MODIFYCTE | VOLATILE)
                            }
                        }
                        IGNORESIDE:true
                    })
                }) && ($orther_expr: (find_expr_without $and_exp $eq_expr))
    CONSTRAINTS: (check_has_constraint ROOT SUBLINK) &&
            (check_type_foreach $exprs AND)
            && (check_type_foreach $FILTER1 AND) && (check_type_foreach $FILTER2 AND) &&
            (check_is_same_table $sub_l_t $sub_r_t)
            && !(check_has_constraint ROOT MODIFYCTE | VOLATILE)
}
=>
QUERY <> {
    FILTER: (append $orther_expr NOT<>{
        ARGS: SUBLINK<existsSUBLINK>{
            QUERY: QUERY<SELECT>{
                PROJECT: CONST<INTEGER>{ARGS:1}
                FROM: (build_fromexpr_by_query $ori_query $input1)
                FILTER: AND <> { ARGS : (append $same_set NOT<>{
                    ARGS: $orther_f
                })
                }
                GROUP: GROUP<> {
                GROUP: (build_group $att_list)
                FILTER: $h_c1
                GROUPSET: $gr_s1
                }
                SORT: $s1
                }
            }
        }
    )
}
```

DSL Rule Analysis

- Matched part:
  - **QUERY<>** indicates that the default query type SELECT is matched.
  - **FILTER<>** indicates that constraint conditions are matched.
  - **ARGS:** *$exprs: ** indicates that all content of the constraint conditions is mapped to the variable *exprs*. **&&** indicates that all conditions must be matched.
  - **AND<>** indicates that only the AND condition is matched.
  - **ARGS:** *$and_exp: ** indicates that all content of the matched AND expression is mapped to the variable *and_expr*. **ARGS: OPERATOR<EQ>** indicates that the equality expression is matched.
  - **LEFT: SUBLINK<EXPRSUBLINK>** and **RIGHT: SUBLINK<EXPRSUBLINK>** indicate that the left and right values of the equality expression must be sublinks, and **PROJECT:** *$proj1*: **AGG<COUNT>{}** in the sublinks indicates that the projection column must be COUNT().
  - ((check_has_constraint $f_1_l UPPERVAR) && !(check_has_constraint $f_1_r UPPERVAR)) ||

    (!(check_has_constraint *$f_1_l* **UPPERVAR) && (check_has_constraint** *$f_1_r* **UPPERVAR))** indicates that the constraint checks for upper-layer variables, that is, the correlation conditions, on one side of the equality expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **GROUP** or **SORT** matches the GROUP or SORT parts in sublinks.
  - **AND<> {ARGS:** *$f_2* **: * && (***$orther_f*: **SUBLINK<EXISTSSUBLINK>{QUERY:** *$ssub_query*: ***})}** indicates that the AND condition of the constraint in the right sublink contains an EXISTSSUBLINK.
  - **$same_set : (find_expr_without** *$f_2 $orther_f***)** indicates that the *$orther_f* part is excluded from *$f_2* and the other part is mapped to the variable *$same_set*.
  - **IGNORESIDE:true** indicates that the values on the left and right sides can be exchanged for template matching.
  - **CONSTRAINTS: (check_is_equal** *$same_set $f_1***)** indicates that the two sublinks are the same except an EXISTSSUBLINK on the right.
  - **(***$orther_expr***: (find_expr_without** *$and_exp $eq_expr***))** indicates that the unmatched part is extracted from the constraint.
  - **(((check_type_foreach** *$proj1* **COUNT_STAR) && (check_type_foreach** *$proj2* **COUNT_STAR)) || ((check_type_foreach (expr_compute** *$agg_args1***) CONST) && (check_type_foreach (expr_compute** *$agg_args2***) CONST)) || ((check_type_foreach** *$agg_args1* ***COL) && (check_type_foreach*** *$agg_args2* **COL) && (check_is_equal** *$agg_args1 $agg_args2***)))** indicates that the parameters of the COUNT() function in the left and right sublinks can only be COUNT(*), COUNT(constant), or COUNT on the same columns.

- Rewritten part:
  - **QUERY<>** is the same as that in the original syntax.
  - **Filter<>** is the same as that in the original syntax.
  - **ARGS** indicates a member variable. **append**, which is an internal function, combines two expressions into a linked list. **append $orther_expr NOT<>** indicates that the expression mapped by *$orther_expr* and the expression following NOT are combined.
  - **ARGS: SUBLINK<EXISTSSUBLINK>** indicates an EXISTS sublink under the NOT node.
  - **GROUP** or **SORT** constructs the GROUP or SORT part in sublinks.
  - **QUERY: QUERY<SELECT>** indicates that the QUERY member of the sublink is an expression.
  - **PROJECT: CONST<INTEGER>{ARGS:1}** indicates that the number of projection columns is 1.
  - **FROM: (build_from_expr_from_query** *$ori_query $input1***)** indicates that the FROM part is constructed by calling the build_from_expr_from_query function.
  - **FILTER: AND <> { ARGS : (append** *$same_set* **NOT<>:{ARGS:** *$orther_f***}** indicates that the filter condition is an AND expression. The expression content is constructed by combining the expression mapped by *$same_set*, the expression mapped by *$orther_f*, and NOT by calling the append function.

Execution Result

```
EXPLAIN (costs off)SELECT t1.C, t1.b, t2.b FROM t1, t2 WHERE t1.a = t2.a AND (SELECT COUNT(*) FROM t1
as t11 WHERE t11.a = t1.a AND t11.b = t1.b) = (SELECT COUNT(*) FROM t1 AS t21 WHERE t21.a = t1.a
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
AND t21.b = t1.b AND EXISTS(SELECT 1 FROM t3 WHERE t1.a = t3.a));
NOTICE:  Query with blockname(sel$1) is rewritten by Gs_Rewriter: SELECT t1.c, t1.b, t2.b FROM dsl_exprsublink_elimination.t1, dsl_exprsublink_elimination.t2 WHERE t1.a = t2.a AND NOT (EXISTS (SELECT 1::bigint FROM dsl_exprsublink_elimination.t1 t11 WHERE t11.a = t1.a AND t11.b = t1.b AND NOT (EXISTS (SELECT 1 AS "?column?" FROM dsl_exprsublink_elimination.t3 WHERE t1.a = t3.a)))); with rule_name sublink_compare_elimination
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Hash Join
   Hash Cond: (t2.a = t1.a), (Expression Flatten Optimized)
   ->  Seq Scan on t2
   ->  Hash
         ->  Merge Anti Join
               Merge Cond: ((t1.a = t11.a) AND (t1.b = t11.b)), (Expression Flatten Optimized)
               Join Filter: (NOT (alternatives: SubPlan 1 or hashed SubPlan 2)), (Expression Flatten Optimized)
               ->  Sort
                     Sort Key: t1.a, t1.b
                     ->  Seq Scan on t1
               ->  Sort
                     Sort Key: t11.a, t11.b
                     ->  Seq Scan on t1 t11
               SubPlan 1
                 ->  Seq Scan on t3
                       Filter: (t1.a = a), (Expression Flatten Optimized)
               SubPlan 2
                 ->  Seq Scan on t3
(18 rows)
```

7.15.3 Best Practices for the DSL Rule Engine

To use the DSL rule engine, you must set GUC parameter **enable_dsl_ruleengine** to **ON**.

The following provides best practice examples.

NOTE

For details about the permissions to call system functions in the best practices, see the following sections:

- Non-multi-primary mode: See "SQL Reference > Functions and Operators > DSL Rule Management Functions" in *Reference*.

- Scenario 1: Create a DSL rule by calling the dsl_add_rule function as an authorized user.

```
gaussdb=# SELECT dsl_add_rule(
'compute_const',
'QUERY<> {FILTER: $expr: *}
=>
QUERY<> {FILTER: (expr_compute $expr)}
',
777,
true);
 dsl_add_rule
---------------
 compute_const
(1 row)

-- Load the new rule so that it can be applied.
gaussdb=# SELECT dsl_reload_rule();
 dsl_reload_rule
-----------------
 success
(1 row)
```

- Scenario 2: Run SQL statements as an authorized user to query the DSL rules in the system catalog.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Switch to the extended display mode to facilitate plan observation.
gaussdb=# \x
Expanded display is on.

gaussdb=# SELECT * FROM GS_DSL_REWRITERULE WHERE rulename = 'compute_const';
-[ RECORD 1 ]---+---------------------------------------
rulename        | compute_const
ruledesc        |
enabled         | t
externalrule    | t
costbased       | f
priority        | 777
dbcompatibility | 31
planmode        | 3
createtime      | xxxx-xx-xx xx:xx:xx.596976+08
lastupdatetime  |
rulehash        | xxxxxxxxxxxxxxxxx
ruletext        | QUERY<> {FILTER: $expr: *}
                | =>
                | QUERY<> {FILTER: (expr_compute $expr)}
                |
```

- Scenario 3: Modify a DSL rule in the system catalog by calling the dsl_set_rule function as an authorized user.

```
-- Set rule_name to compute_const and priority to 8.
gaussdb=# SELECT dsl_set_rule('compute_const', 'priority', '778');
-[ RECORD 1 ]+--
dsl_set_rule | 1

gaussdb=# SELECT * FROM GS_DSL_REWRITERULE WHERE rulename = 'compute_const';
-[ RECORD 1 ]---+---------------------------------------
rulename        | compute_const
ruledesc        |
enabled         | t
externalrule    | t
costbased       | f
priority        | 778 -- The priority of the rule is set to 8.
dbcompatibility | 31
planmode        | 3
createtime      | xxxx-xx-xx xx:xx:xx.596976+08
lastupdatetime  | xxxx-xx-xx xx:xx:xx.822819+08
rulehash        | xxxxxxxxxxxxxxxxxxxx
ruletext        | QUERY<> {FILTER: $expr: *}
                | =>
                | QUERY<> {FILTER: (expr_compute $expr)}
                |
```

- Scenario 4: Apply a rule to rewrite SQL statements that meet the necessary conditions.

```
-- Enable the DSL rule engine.
gaussdb=# SET enable_dsl_ruleengine = ON;
SET
gaussdb=# SHOW enable_dsl_ruleengine;
 enable_dsl_ruleengine
-----------------------
 on
(1 row)
-- Enable log printing for DSL rules.
gaussdb=# set logging_module = 'on(OPT_DSL_REWRITE)';
SET

-- Disable the extended display mode.
gaussdb=# \x
Expanded display is off.

gaussdb=# EXPLAIN(VERBOSE ON) SELECT * FROM t1 WHERE t1.a = 1 OR TRUE;
NOTICE:  Query with blockname(sel$1) is rewritten by Gs_Rewriter: SELECT a FROM public.t1 WHERE
true; with rule_name compute_const
                        QUERY PLAN
------------------------------------------------------------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 [Parameterized]
 Seq Scan on public.t1  (cost=0.00..34.02 rows=2402 width=4)
   Output: a
(3 rows)
```

- Scenario 5: Delete an existing DSL rule by calling the dsl_del_rule function as an authorized user and specifying the rule name.

```
gaussdb=# SELECT dsl_del_rule('compute_const');
 dsl_del_rule
--------------
            1
(1 row)
-- Delete a rule from the cache.
gaussdb=# SELECT dsl_reload_rule();
 dsl_reload_rule
-----------------
 success
(1 row)
```

7.15.4 Troubleshooting

ERROR: syntax error at 'XXXX'.

- Scenario 1: The keyword is incorrect.

```
gaussdb=# SELECT dsl_add_rule(
'compute_const',
'JOIN<> {ARGS: $expr: *}
=>
FILTER<> {ARGS: (expr_compute $expr)
}',
3,
true);
ERROR:  syntax error at 'JOIN'  --
LINE 3: 'JOIN<> {ARGS: $expr: *}
        ^
CONTEXT:  referenced column: dsl_add_rule
```

  Root cause: The JOIN keyword cannot be used as the first keyword regarding this syntax. For details about how to use keywords, see **Keywords**.

- Scenario 2: Parentheses (()) are not used as separators in the function.

```
gaussdb=# SELECT dsl_add_rule(
'compute_const',
'FILTER<> {ARGS: $expr: *}
=>
FILTER<> {ARGS: expr_compute $expr
}',
3,
true);
ERROR:  syntax error at 'expr_compute'
LINE 5: FILTER<> {ARGS: expr_compute $expr
                        ^
CONTEXT:  referenced column: dsl_add_rule
```

  Root cause: The function keyword and its input parameter variables are not separated by parentheses (()). For details about how to use functions, see **Functions**.

ERROR: Duplicate key value violates unique constraint "gs_dsl_rewriterule_rulename_index".

Scenario: The rule name is duplicate.

```
-- Add a rule whose rulename is compute_const.
gaussdb=# SELECT dsl_add_rule(
'compute_const',
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
'FILTER<> {ARGS: $expr: *}
=>
FILTER<> {ARGS: (expr_compute $expr)
}',
3,
true);
 dsl_add_rule
---------------
 compute_const
(1 row)

-- Add a rule whose rule name is compute_const again.
gaussdb=# SELECT dsl_add_rule(
gaussdb(# 'compute_const',
gaussdb(# 'FILTER<> {ARGS: $expr : *}
gaussdb'# =>
gaussdb'# FILTER<> {ARGS: (expr_compute $expr)
gaussdb'# }',
gaussdb(# 3,
gaussdb(# true);
ERROR:  Duplicate key value violates unique constraint "gs_dsl_rewriterule_rulename_index".
DETAIL:  Key (rulename)=(compute_const) already exists.
CONTEXT:  referenced column: dsl_add_rule
```

Root cause: The value of **rulename** must be globally unique. For details about the restrictions, see the following sections:

- Non-multi-primary mode: See "System Catalogs and System Views > System Catalogs > DSL Rule Engine" in *Reference*.

7.16 Foreign Data Wrapper

Foreign data wrappers (FDWs) of GaussDB can implement cross-database operations between GaussDB databases and remote servers (including databases and file systems). Currently, the supported FDW is file_fdw.

7.16.1 file_fdw

The file_fdw module provides the foreign data wrapper file_fdw, which can be used to access data files in the file system of a server. The data file must be readable by COPY FROM. For details, see "SQL Reference > SQL Syntax > C > COPY" in *Reference*. file_fdw is only used to access readable data files, but cannot write data to the data files.

By default, file_fdw is compiled in GaussDB. During database initialization, the plug-in is created in the pg_catalog schema.

The server and foreign table corresponding to file_fdw can be created only by the initial user of the database, system administrators, or O&M administrators when the O&M mode is enabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- To prevent arbitrary read of files on the server, system administrators and O&M administrators in O&M mode are subject to the **enable_copy_server_files** and **safe_data_path** parameters.
- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to create foreign tables of the file_fdw type. When this parameter is enabled, system administrators and O&M administrators in O&M mode can create foreign tables of the file_fdw type to prevent unauthorized users from viewing or modifying sensitive files.
- When the **enable_copy_server_files** parameter is enabled, administrators can use the GUC parameter **safe_data_path** to set the path for users to import and export. The files must be stored in a subpath of the set path. If this GUC parameter is not set (by default), the path used by users is not blocked.

When you create a foreign table using file_fdw, you can add the following options:

- filename

  File to be read. This parameter is required and must be an absolute path.

- format

  File format of the remote server, which is the same as the **FORMAT** option of the COPY statement. The value can be **text**, **csv**, or **binary**. This parameter is optional.

- header

  Specifies whether a specified file has a header, which is the same as the **HEADER** option of the COPY statement. This parameter is optional.

- delimiter

  File delimiter, which is the same as the **DELIMITER** option of the COPY statement. This parameter is optional.

- quote

  Quote character of a file, which is the same as the **QUOTE** option of the COPY statement. This parameter is optional.

- escape

  Escape character of a file, which is the same as the **ESCAPE** option of the COPY statement. This parameter is optional.

- null

  Null string of a file, which is the same as the **NULL** option of the COPY statement. This parameter is optional.

- encoding

  Encoding of a file, which is the same as the **ENCODING** option of the COPY statement. This parameter is optional.

- force_not_null

  This is a Boolean option and is an optional parameter. If it is true, the value of the declared column cannot be a null string (that is, file-level **null** option). This option is the same as the **FORCE_NOT_NULL** option of the COPY statement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- file_fdw does not support the **OIDS** and **FORCE_QUOTE** options of the COPY statement.
- These options can only be declared for a foreign table or the columns of the foreign table, not for file_fdw itself, nor for the server or user mapping that uses file_fdw.
- To modify table-level options, you must obtain the SYSADMIN permissions. For security reasons, only system administrators can determine the files to be read.
- For a foreign table that uses file_fdw, running **EXPLAIN** displays the name and size (in bytes) of the file to be read. If the keyword **COSTS OFF** is specified, the file size is not displayed.

Using file_fdw

- To create a server object, run **CREATE SERVER**.
- To create a user mapping, run **CREATE USER MAPPING**.
- To create a foreign table, run **CREATE FOREIGN TABLE**.

  NOTE

  - The structure of the foreign table must be consistent with the data in the specified file.
  - When a foreign table is queried, write operations are not allowed.
- To drop a foreign table, run **DROP FOREIGN TABLE**.
- To drop a user mapping, run **DROP USER MAPPING**.
- To drop a server object, run **DROP SERVER**.

Precautions

- To use file_fdw, you need to specify the file to be read. Prepare the file and grant the read permission on the file for the database to access the file.
- DROP EXTENSION cannot be used for file_fdw.

7.17 GTM

7.17.1 Overview

GTM Overview

A Global Transaction Manager (GTM) distributes transaction IDs (XIDs), snapshots, and sequence numbers to ensure global transaction consistency.

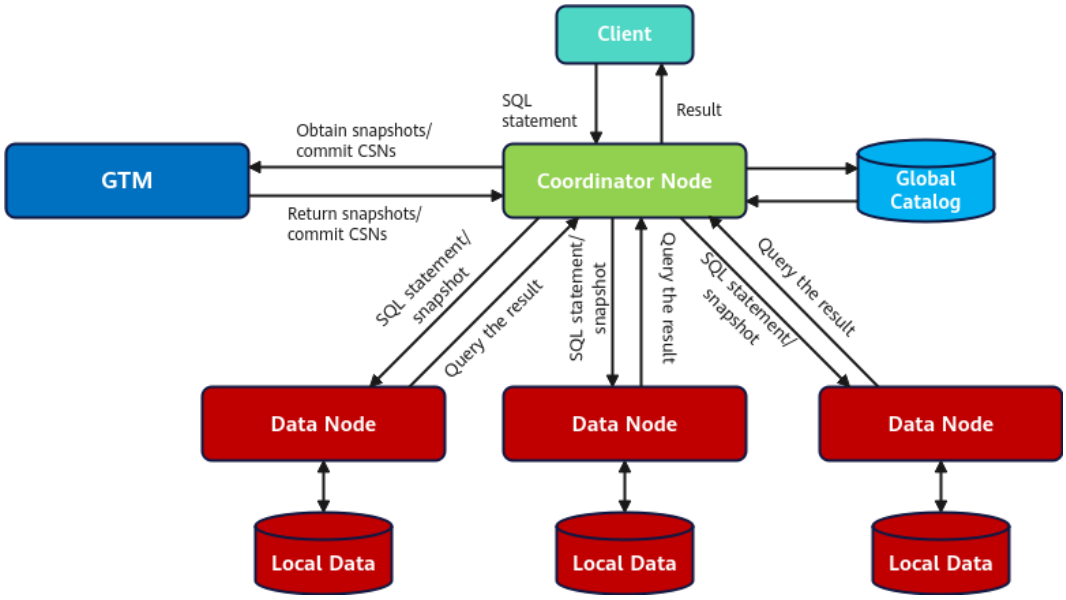
Technical Architecture

A GTM works in primary/standby mode. It is a resident service process in the system and uses the multi-thread architecture. It provides three main functions: generating XIDs, generating snapshots, and managing sequence numbers.

A GTM passively accepts connections. Generally, only CNs actively connect to the GTM, and DNs connect to the GTM when autovacuum (background tuple clearing thread) is working. **Figure 7-36** shows the relationship between a GTM and the entire HA architecture.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-36 GTM architecture



NOTE

In GTM-free mode, CNs do not obtain snapshots or commit CSNs via the GTM. For details, see **GTM-Free**.

Three GTM states:

Pending: The GTM is initially started, that is, the GTM is to be arbitrated.

Primary: The GTM is the primary one.

Standby: The GTM is the standby one.

Two GTM connection states:

Connection ok: The primary and standby nodes are properly connected.

Connection bad: The primary and standby nodes are not properly connected.

GTM synchronization states:

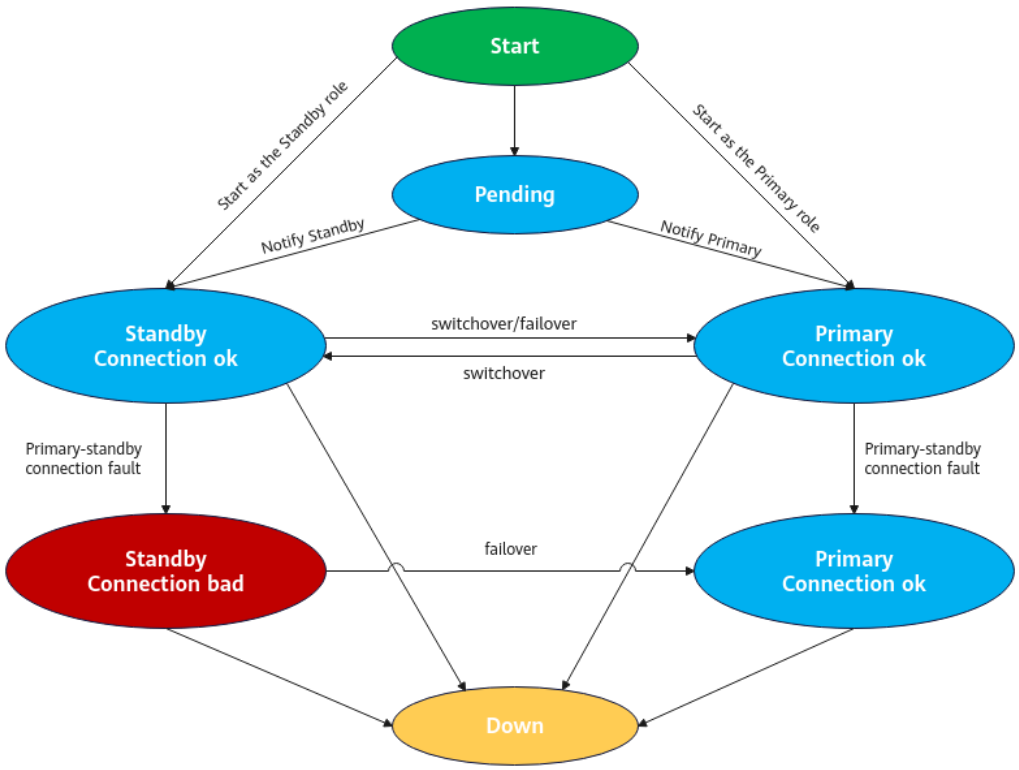
Sync: synchronization mode (The primary node needs to wait for the synchronization of the standby node when writing data.)

Most available: maximum availability mode (The primary node does not need to wait for the synchronization of the standby node when writing data.)

Figure 7-37 shows the three GTM states and two connection states.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-37 GTM status change



NOTE

In cluster deployment, the CM starts a GTM component in **Pending** mode and then instructs the GTM to be promoted to primary or demoted to standby after primary/standby arbitration. You can use the gtm_ctl tool to start a GTM in a specified mode. However, exercise caution when performing this operation. You are advised not to directly perform operations on the GTM status. The GTM status should be managed by the CM.

7.17.2 GTM Modes

To meet different concurrency and consistency requirements, GaussDB provides three GTM modes: GTM, GTM-lite, and GTM-free. The main differences between the two modes lie in the pressure of the central transaction management node GTM and the transaction processing process.

The current version does not support the switchover between the three modes. You are advised to use the default GTM-lite mode during installation, so that the GTM mode can remain unchanged before and after the upgrade.

7.17.2.1 GTM

7.17.2.1.1 GTM Mode

Description

The GTM mode uses the global transaction ID policy. All nodes need to obtain the globally incremental transaction IDs (CSNs) through the GTM, and obtaining the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

CSNs through the GTM is also essential for ensuring the consistency of committed distributed transactions.

Scenarios

The current version no longer supports GTM mode. If you need to ensure global transaction consistency, GTM-lite mode is a better choice.

Specifications and Limitations

The GTM mode is not supported in version 503.0.0 and later. If the GTM mode is configured, the GTM-lite mode still takes effect.

7.17.2.1.2 Technical Principles

In GTM mode, a GTM is responsible for maintaining global transaction management and increment of global CSNs and sequence numbers.

When a transaction is started on an execution CN, the CN needs to apply for a globally incremental transaction ID from the GTM to ensure the consistency of distributed transactions. When the transaction is committed, the CN obtains the global CSN from the GTM as the transaction CSN.

7.17.2.2 GTM-Lite

7.17.2.2.1 GTM-Lite Mode

Description

In the case of strong consistency of distributed transactions, GTM-lite mode is the optimal deployment mode. It converts global transaction IDs to local transaction IDs and maintains global CSNs. This ensures strong consistency of global transactions and mitigates single-point performance bottlenecks.

In GTM mode, the global transaction ID policy is used. Therefore, each active transaction must be registered on a GTM. The GTM manages and distributes global transaction IDs. As a result, when the service concurrency is high, the GTM is likely to incur a single-point performance bottleneck.

Unlike GTM mode, in GTM-free mode, CNs and DNs do not interact with GTMs. Local transaction IDs on CNs and DNs are maintained respectively to ensure the normal running of the transaction system. In this mode, transaction consistency is affected due to the lack of global transaction IDs and other GTM coordination (CSN). As a result, the application scope of this mode is limited.

In conclusion, GTM and GTM-free modes do not achieve a good balance between efficiency and consistency. The main factor that affects the balance is coordination. Coordination ensures more consistency, but reduces performance. Therefore, GTM-lite mode achieves the highest possible performance with the least possible coordination.

Scenarios

Currently, GTM-lite mode is the mainstream application scenario. It is the optimal choice for scenarios where users need to ensure global transaction consistency,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

and where the performance is optimal and the concurrency bottleneck is the lowest.

7.17.2.2.2 Technical Principle

The main objective of GTM-lite mode is to eliminate the impact of GTM bottlenecks, ensure transaction consistency, and improve performance by minimizing information exchange and coordinating transaction concurrency. Therefore, the technical principle of GTM-lite mode includes the following points:

- Local transaction ID: Transaction IDs are no longer obtained from the GTM. Each CN/DN maintains its own transaction ID.
- Global CSN: A GTM maintains only the global CSNs and no longer maintains transaction slot information.
- Single-node transaction optimization: Multi-version prepared arrays (two-phase transaction arrays) are introduced to optimize the visibility determination of single-node transactions.
- Local multi-version snapshot recycling: The "Snapshot is invalid" mechanism is introduced. That is, the current snapshot is smaller than the current global minimum CSN. This mechanism solves the problem that the RecentGlobalXmin (minimum global transaction ID used recently) of each node is updated after the global transaction ID is not used, supporting the recycling of local multi-version snapshots.

7.17.2.3 GTM-Free

7.17.2.3.1 GTM-Free Mode

Description

In GTM-free mode, no GTM participates in transaction processing. This eliminates single-point performance bottlenecks caused by GTMs. Using this mode enables higher transaction processing performance. However, in terms of consistency, this mode only supports external consistency of reads after all transactions are completed. It does not support strong-consistency read of distributed transactions, or transaction consistency that depends on query results as follows:

```
INSERT INTO SELECT * FROM *;
```

Scenarios

GTM-free mode is applicable to scenarios where services do not have strong consistency requirements on distributed transactions and user tables use proper distribution keys.

Specifications and Limitations

In GTM-free mode, eventual consistent execution and concurrency control are used for distributed transactions. Therefore, the usage scenarios and methods of some syntaxes are restricted. If the restricted syntaxes are required by services, refactoring is required based on the understanding of the eventual consistent behavior. The involved restricted syntaxes and refactoring suggestions are as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1. General principle: Specify a proper distribution key (by using **DISTRIBUTE BY**) for all user tables.
   - You are advised to distribute data evenly.
   - You are advised to use the join condition in a query as the distribution key to ensure that the join query does not cause data flow between DNs.
   - You are advised to select the primary key of the table as the distribution key.
2. SELECT:
   - During query, the WHERE condition must contain the equivalent query condition of all distribution keys.
   - Do not use subqueries in the SELECT target columns. Otherwise, the plan may fail to be pushed down to DNs for execution, affecting the execution performance.
3. DML:

   By default, cross-node transactions are not supported. If an executed DML statement contains cross-node transactions, an error is reported. There are two scenarios:

   a. If a user statement is split into multiple independent statements in a database for execution, the following error message is displayed:
   ```
   INSERT/UPDATE/DELETE/MERGE contains multiple remote queries under GTM-free mode
   Unsupport DML two phase commit under gtm free mode. modify your SQL to generate light-proxy or fast-query-shipping plan.
   ```
   In this case, you need to modify the statements and execute them respectively on a single node. For example:
   ```
   INSERT INTO t SELECT * FROM b WHERE b.c = xx;
   ```
   Assume that the distribution keys of tables **t** and **b** are different, and the WHERE condition filters out only one data record. If **enable_stream_operator** is disabled, the preceding query is split into two independent statements and executed in serial mode. First, run **SELECT * FROM b WHERE b.c = xx** to extract data from a DN to the target record. Then, run **INSERT INTO t** to deliver the extracted target record to another DN for insertion. In GTM-free mode, the preceding error is reported when such a statement is executed. Similarly, errors may occur in statements such as CREATE TABLE AS SELECT * FROM and DELETE/JOIN/INSERT with subqueries.

   Refactoring solution: Before executing a service statement, you need to run the following command to enable the Stream operator so that the statement can be pushed down and executed as a whole:
   ```
   SET enable_stream_operator=on;
   ```
   b. If a user statement is executed on multiple nodes in the database, the following error is reported:
   ```
   Your SQL needs more than one datanode to be involved in.
   ```
   In this case, you are advised to modify the statement so that it can be executed on a single node. For example:
   ```
   INSERT INTO t VALUES(3,3),(1,1);
   ```
   If (3,3) and (1,1) are distributed on different DNs, the execution of the preceding statement involves two DNs. In GTM-free mode, the preceding error is reported when such a statement is executed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Refactoring solution: If the preceding statement needs to be executed on multiple nodes, add a hint to the statement to prevent errors. The hint is as follows:

```
INSERT /*+ multinode */ INTO t VALUES(3,3),(1,1);
```

There are similar constraints on the DELETE and UPDATE statements. You are advised to add the equality filter condition of the distribution key to the WHERE condition in the DELETE and UPDATE statements.

4. It is recommended that **application_type** be set to **perfect_sharding_type** in the JDBC connection string in the development phase. In this way, errors are reported for all SQL statements for cross-node read and write operations, prompting developers to optimize statements as soon as possible.

7.17.2.3.2 Technical Principles

In GTM-free mode, each node uses its own transaction ID and local snapshot CSN, eliminating the dependency of transactions on a GTM.

Therefore, the consistency of distributed transactions is not strongly guaranteed.

7.17.3 Operation Guide

Setting GTM Mode

Related GUC parameters include **enable_gtm_free** and **gtm_option**. (For details, see "Database GUC Parameters > GUC Parameters > GTM Parameters" in *Reference*.) You can run the following SHOW statement using gsql to query the current GTM mode:

```
SHOW enable_gtm_free;
SHOW gtm_option;
```

The method of determining the mode is as follows:

- GTM mode: **enable_gtm_free=off** and **gtm_option=0** (It is not supported in the current version.)
- GTM-lite mode: **enable_gtm_free=off** and **gtm_option=1**
- GTM-free mode: **enable_gtm_free=on** or **gtm_option=2**

You can also set GTM mode by referring to the preceding content.

NOTE

GTM-lite mode reduces the single-point performance pressure under the strong consistency of distributed transactions. The concurrency performance of transactions in GTM-lite mode is better than that in GTM mode. GTM mode is not supported in the current version. If **enable_gtm_free** is set to **off** and **gtm_option** is set to **0**, GTM-lite mode is still used.

7.17.4 Best Practices for Distributed Mode

GTM deployment is closely related to service requirements and cluster scale.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.17.5 Typical Issues

Issue 1: SQL Statement Execution Is Slow, and GTM-related WARNING Messages Are Displayed in Logs

Analysis: When executing an SQL statement, the system needs to obtain a snapshot from the GTM to determine the transaction visibility. The following alarm is displayed in the log, indicating that the GTM fails to be connected:

```
Can not connect to all GTMs and then retry(x times).
```

Solution: You can run the following command to query the GTM status:

```
cm_ctl query -Cvip
```

- If no primary GTM exists, GTMs are being switched over. Wait until the switchover is complete.
- If a primary GTM exists and is normal, the network communication between nodes may be faulty. Run the **ping** command to check whether the network communication between the CN and the primary GTM is normal.

Issue 2: The Maximum Time Consumed by the Wait Event of Slow SQL Diagnosis is gtm get snapshot

Analysis: After slow SQL statements are detected, query the maximum **wait_event** duration from the statement_history view (For details, see "System Catalogs and System Views > System Catalogs > Other System Catalogs > STATEMENT_HISTORY" in *Reference*) and find that the result is **gtm get snapshot**.

Solution: Run the **ping** command to check whether the network latency between the CN and the primary GTM is normal.

7.18 Sensitive Data Discovery

The sensitive data discovery functionality provides the gs_sensitive_data_discovery() and gs_sensitive_data_discovery_detail() functions. By calling different functions, you can specify the scan object and sensitive data classifier to obtain sensitive data of different levels corresponding to the scan object.

Using the Sensitive Data Discovery Functions to Scan Data

Step 1 Create a scan target.

```
gaussdb=# CREATE SCHEMA test_schema1;
gaussdb=# CREATE TABLE test_schema1.normal_table1 (email VARCHAR(50), phonenum text, creditcard bytea, name BPCHAR(50), encrypted_data NVARCHAR2(500));
gaussdb=# INSERT INTO test_schema1.normal_table1 VALUES ('Gauss_1@huawei.com', '13345678901', '4951-7843-2960-0886', 'GaussDB',  'r1Pj4DKQopSmZfSr/mXwieGcrE+O2iX3G04w1Tda0zqX/pD/Az90tLuvy0ZnCmUg0/Kl/g89ZqxxAZzta8jQ6bvicRpdXrDCQTHAfsXYSEScB9Idqtwh7kdm1LHq89qs');
```

Step 2 Call the sensitive data discovery function to scan the target.

- Call the gs_sensitive_data_discovery() function to scan the target data and return the scan result.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# SELECT * FROM gs_sensitive_data_discovery('test_schema1.normal_table1',  'all');
           target           | percentage_of_sensitive_data |
classifier_type_distribution
----------------------------+------------------------------
+-----------------------------------------------------------------------------------------
 test_schema1.normal_table1 | 5/5                          |  Email: 1/5  CreditCard: 1/5  PhoneNumber: 1/5
ChineseName: 1/5  EncryptedContent: 1/5
(1 row)
```

- Call the gs_sensitive_data_discovery_detail() function to scan the target data and return the detailed scan result.

```
gaussdb=# SELECT * FROM gs_sensitive_data_discovery_detail('test_schema1',  'all');
   schema    |     table     |    column      |    classifier    | laws_and_regulations
-------------+---------------+----------------+------------------+----------------------
 test_schema1 | normal_table1 | email          | Email            | GDPR
 test_schema1 | normal_table1 | phonenum       | PhoneNumber      | GDPR
 test_schema1 | normal_table1 | creditcard     | CreditCard       | GDPR
 test_schema1 | normal_table1 | name           | ChineseName      | GDPR
 test_schema1 | normal_table1 | encrypted_data | EncryptedContent | GDPR
(5 rows)
```

Step 3 Clear data.

```
gaussdb=# DROP TABLE test_schema1.normal_table1;
gaussdb=# DROP SCHEMA test_schema1;
(3 row)
```

----End

7.19 Dynamic Data Masking

Data masking is an effective database privacy protection solution, which can prevent attackers from snooping on private data. The dynamic data masking mechanism is a technology that protects privacy data by customizing masking policies. It can effectively prevent unauthorized users from accessing sensitive information while retaining original data. After the administrator specifies the object to be masked and customizes a data masking policy, if the database resources queried by a user are associated with a masking policy, data is masked based on the user identity and masking policy to restrict attackers' access to privacy data.

Viewing the Basic Configurations of Dynamic Data Masking

Run the following command to check whether dynamic data masking is enabled:

```
gaussdb=# SHOW enable_security_policy;
 enable_security_policy
------------------------
 on
(1 row)
```

If **enable_security_policy** is set to **on**, the function is enabled. If it is set to **off**, the functionality is disabled. If the value is **off**, set the parameter by referring to "Database GUC Parameters > GUC Parameters > Security Configuration" in *Reference*.

Creating a Masking Policy

Step 1 Create a table.

```
-- Create table tb_for_masking.
gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text, col6
text, col7 text,col8 text);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Insert data into the tb_for_masking table.
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@example.com',
'abc@example.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3', '4880-9898-4545-2525',
'this is a llt case');

-- View data.
gaussdb=# SELECT * FROM tb_for_masking;
 idx |    col1    |   col2    |      col3      |      col4      |        col5         |              col6
     |        col7         |        col8
-----+------------+-----------+----------------+----------------+---------------------+-------------------------------
------+---------------------+--------------------
   1 | 9876543210 | usr321usr | abc@example.com | abc@example.com | 1234-4567-7890-0123 | abcdef
123456 ui 323 jsfd321 j
3k2l3 | 4880-9898-4545-2525 | this is a llt case
(1 row)
```

Step 2 Create a resource label.

```
-- Create a resource label for sensitive column col1.
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);
CREATE RESOURCE LABEL
gaussdb=# CREATE RESOURCE LABEL mask_lb5 ADD COLUMN(tb_for_masking.col5);
CREATE RESOURCE LABEL
```

Step 3 Create a masking policy.

For details about the syntax format, see "SQL Reference > SQL Syntax > C > CREATE MASKING POLICY" in *Reference*.

```
-- Create a data masking policy named maskpol1.
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);
CREATE MASKING POLICY
```

During the configuration of dynamic data masking policies, custom functions created by users can be adapted.

```
gaussdb=# create or replace function msk_creditcard(col text) returns TEXT as $$
declare
    result TEXT;
begin
    result := overlay(col placing 'xxxx-xxxx' from 6);
    return result;
end;
$$ language plpgsql security DEFINER;
CREATE FUNCTION
-- Create a data masking policy named maskpol5.
gaussdb=# CREATE MASKING POLICY maskpol5 msk_creditcard ON LABEL(mask_lb5);
CREATE MASKING POLICY
```

Step 4 Query the masked column data in the **tb_for_masking** table.

```
-- Access the tb_for_masking table. The masking policy is triggered for the col1 column.
gaussdb=# SELECT col1 FROM tb_for_masking;
    col1
------------
 xxxxxxxxxx
(1 row)
-- Access the tb_for_masking table. The masking policy is triggered for the col5 column.
gaussdb=# SELECT col5 FROM tb_for_masking;
        col5
---------------------
 1234-xxxx-xxxx-0123
(1 row)
```

Step 5 Clear data.

```
-- Drop masking policies.
gaussdb=# DROP MASKING POLICY maskpol1, maskpol5;
DROP MASKING POLICY
-- Drop resource labels.
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb5;
DROP RESOURCE LABEL
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the tb_for_masking table.
gaussdb=# DROP TABLE tb_for_masking;
DROP TABLE
```

----End

7.20 Hybrid Transactional/Analytical Processing

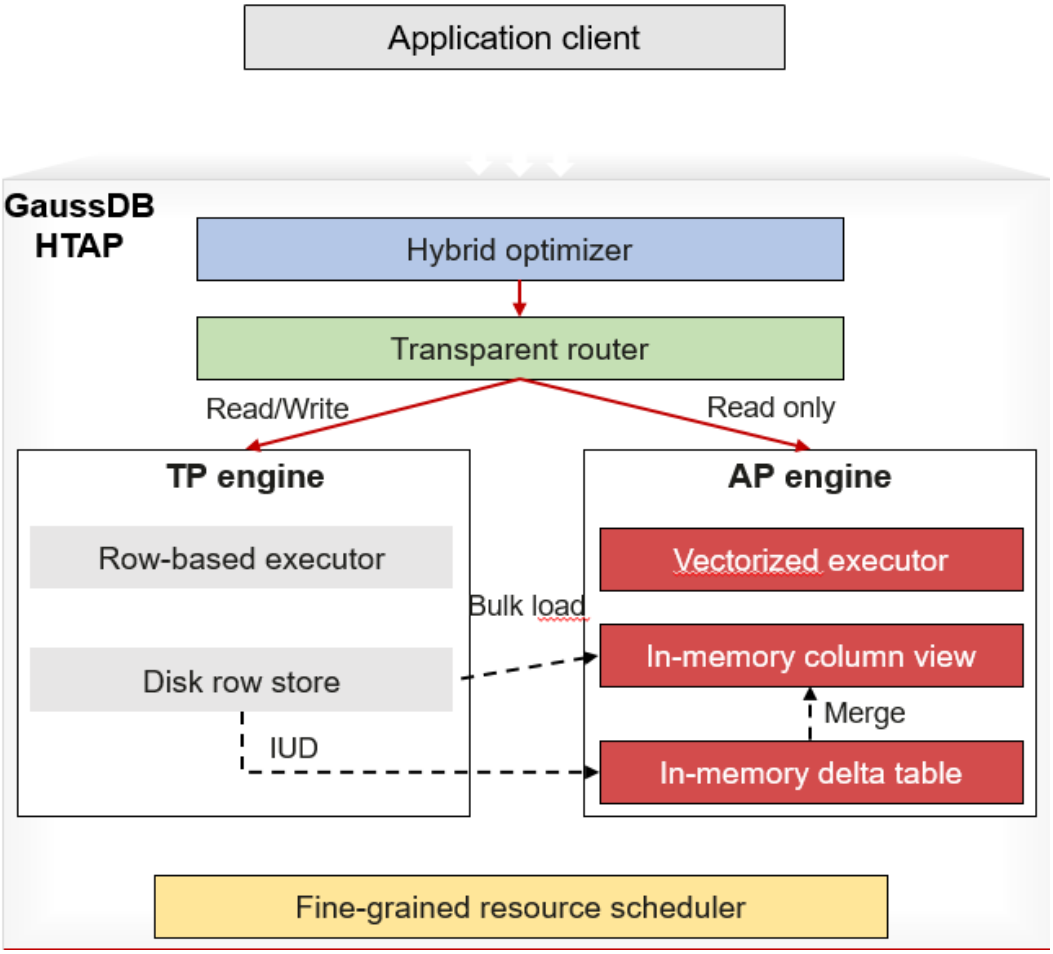
7.20.1 Overview

In addition to using disk row store and volcano model-based executor to efficiently support transaction query loads in traditional transactional databases, HTAP introduces column store, vectorized engine, and transparent router for hybrid loads to efficiently process both transactional and analytical query loads in a single transactional database.

HTAP Architecture

Based on the original row-store data engine and execution engine, HTAP introduces the in-memory column view (IMCV), transparent routing for hybrid loads, and vectorized engine to improve the processing capability of real-time AP queries. For details, see **Figure 7-38**.

Figure 7-38 HTAP architecture



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- IMCV: helps maintain in-memory row-store data snapshots using a columnar format for rapid scan.
- Vectorized executor: batch executes column scans to support faster join, aggregation, and expression evaluation.
- Transaction metadata unit (TMU): records real-time active transaction row updates based on IMCV snapshots and supports IMCV data scan to obtain real-time active data. That is, the IMCV scan result is consistent with the SeqScan result.
- Transparent router for hybrid loads: identifies TP or AP queries and routes them to a row- or column-based engine for efficient execution.
- Resource control and scheduling: implements memory resource control for TP and AP loads in HTAP scenarios.

7.20.2 Usage Guide

By default, the distributed GaussDB is MySQL-compatible. The HTAP functionality supports only ORA-compatible databases. Therefore, you need to create an ORA-compatible database. All examples in this section are performed in this database.

Example

```
gaussdb=# CREATE DATABASE htapdb WITH DBCOMPATIBILITY='ORA';
CREATE DATABASE
gaussdb=# \c htapdb
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "htapdb" as user "xxx".
htapdb=#
```

7.20.2.1 Parameter Settings

- HTAP functionality

  The HTAP functionality is controlled by the **enable_htap** parameter. The value **on** indicates that the functionality is enabled, and **off** indicates that it is disabled. For details about the instance-level parameter **enable_htap**, see "Database GUC Parameters > GUC Parameters > Hybrid Transactional/Analytical Processing" in *Reference*.

- IMCV column-store scan functionality

  The IMCV column-store scan functionality is controlled by the **enable_imcvscan** parameter. The value **on** indicates that the functionality is enabled, and **off** indicates that it is disabled. **enable_imcvscan** is a session-level parameter. You can set it in the configuration file or run **set enable_imcvscan = on** in a session.

- HTAP routing mode setting

  The HTAP routing mode is controlled by the **htap_router_mode** parameter. If this parameter is set to **auto**, the row-store or column-store engine is selected based on the cost calculation during actual execution. **htap_router_mode** is a session-level parameter. You can set it in the configuration file or run **set htap_router_mode = 'auto'** in a session.

  NOTE

  New HTAP parameters can be combined and set based on the actual situation. For details about the new parameters, see "Database GUC Parameters > GUC Parameters > Hybrid Transactional/Analytical Processing" in *Reference*.

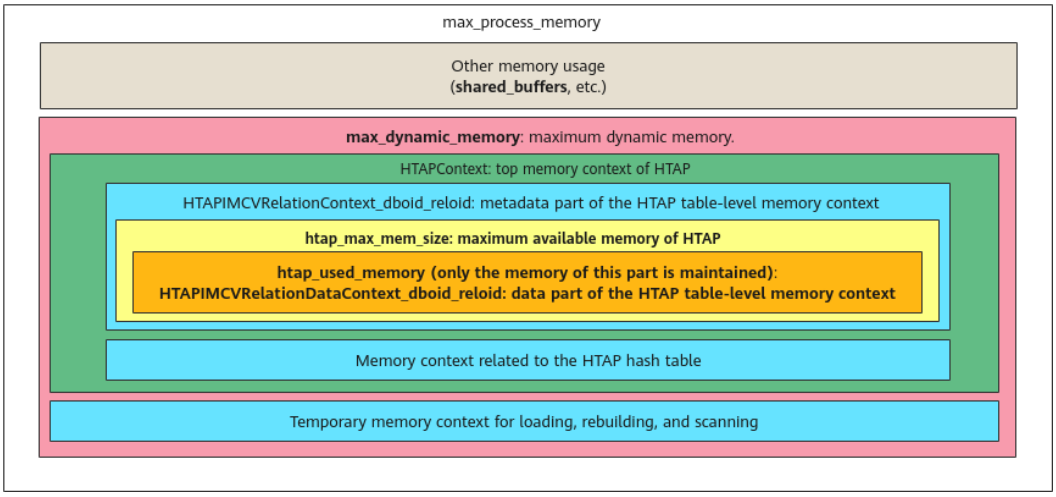
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.20.2.2 Memory Control

All HTAP data is stored in the GaussDB memory. The memory control functionality is configured to prevent row-store query from being affected or database memory overflow due to excessive memory usage.

The HTAP memory consists of dynamic shared memory and non-dynamic shared memory. The non-dynamic shared memory stores temporary memory contexts generated from operations such as loading, rebuilding, and scanning. The dynamic shared memory has three layers: The top-layer dynamic memory stores contexts related to IMCV metadata and auxiliary data required for HTAP, such as hash tables. The second layer is mounted to the top layer of memory contexts, which stores related metadata in each IMCV table, mainly IMCU metadata and TMU data. The third layer stores actual IMCU data, which is mounted to the second layer. **Figure 7-39** shows the memory context structure. The **htap_max_mem_size** parameter is used to monitor the memory overhead of the data part of the HTAP table-level memory context in all HTAP memory. Other parts are regarded as noise and are not monitored.

Figure 7-39 HTAP memory context structure



The related GUC parameters are **htap_max_mem_size** and **htap_memctl_policy**. For details about the GUC parameters, see "Database GUC Parameters > GUC Parameters > Hybrid Transactional/Analytical Processing" in *Reference*.

- **htap_max_mem_size**: specifies the size of available HTAP memory. The value ranges from 100 MB to 1 TB. The default value is **1GB**.
- **htap_memctl_policy**: specifies the memory control policy of the HTAP feature. The value can be **'swap'** or **'escape'** (default).

In the pg_total_memory_detail system view, two records whose **memorytype** is **max_htap_memory** and **htap_used_memory** are added to indicate the maximum available memory and used memory of HTAP, respectively. For details, see "System Catalogs and System Views > System Views > Other System Views > PG_TOTAL_MEMORY_DETAIL" in *Reference*.

You can use the gs_global_imcv_info() system function to confirm the basic IMCV information about a table. For details about this function, see "SQL Reference >

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Functions and Operators > System Administration Functions > HTAP System Functions" in *Reference*.

Examples

```
-- Use a distributed a cluster with 3 CNs and 2DNs, and one primary node and one standby node deployed.
Connect to CN5001 using gsql.
-- View GUC parameters.
htapdb=# SHOW htap_max_mem_size;
 htap_max_mem_size
-------------------
 1GB
(1 row)

htapdb=# SHOW htap_memctl_policy;
 htap_memctl_policy
--------------------
 swap
(1 row)

-- Connect to the primary node of DN6001 and check the memory status and table information of the DN.
htapdb=# SELECT * FROM pg_total_memory_detail WHERE memorytype='max_htap_memory' OR
memorytype='htap_used_memory';
   nodename   |    memorytype    | memorymbytes
--------------+------------------+--------------
 dn_6001_6002 | max_htap_memory  |         1024
 dn_6001_6002 | htap_used_memory |            0
(2 rows)

-- Connect to CN5001.
htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info();
 node_name | rel_name | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem | cu_mem_size |
cu_num_in_disk | cu_disk_size
-----------+----------+-----------+---------+--------------+---------------+-------------+----------------+--------------
(0 rows)

-- Create an ordinary table htap_imcv_mem_ctl_1 and insert a small amount of data into the table.
htapdb=# CREATE TABLE htap_imcv_mem_ctl_1(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) distribute by hash(id);
CREATE TABLE

htapdb=# INSERT INTO htap_imcv_mem_ctl_1 VALUES (generate_series(1,1000000),
generate_series(1,1000000), generate_series(1,1000000), 'Jack', 'AAAA');
INSERT 0 1000000

-- Run the ALTER COLVIEW statement to create an IMCV for the table with a small amount of data.
htapdb=# ALTER TABLE htap_imcv_mem_ctl_1 COLVIEW(id, salary, name, comments);
ALTER TABLE

-- Perform manual loading to ensure that all row-store data has been loaded to the IMCV.
htapdb=# SELECT total from gs_imcv_flush('htap_imcv_mem_ctl_1');
 total
-------
 6
(1 row)

-- Check the memory status and table information on the DN.
htapdb=# EXECUTE DIRECT ON (dn_6001_6002) $$SELECT * FROM pg_total_memory_detail WHERE
memorytype='max_htap_memory' OR memorytype='htap_used_memory'$$;
   nodename   |    memorytype    | memorymbytes
--------------+------------------+--------------
 dn_6001_6002 | max_htap_memory  |         1024
 dn_6001_6002 | htap_used_memory |            3
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info();
  node_name   |      rel_name       | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem | cu_mem_size |
cu_num_in_disk | cu_disk_size
--------------+---------------------+-----------+---------+--------------+---------------+-------------+----------------
+--------------
 dn_6001_6002 | htap_imcv_mem_ctl_1 |         4 | 1 3 4 5 |            3 |            15 |     3553162 |              0
|            0
 dn_6003_6004 | htap_imcv_mem_ctl_1 |         4 | 1 3 4 5 |            3 |            15 |     3545231 |              0
|            0
(2 rows)

- Insert a small amount of data into an existing table with an IMCV.
htapdb=# INSERT INTO htap_imcv_mem_ctl_1 SELECT * FROM htap_imcv_mem_ctl_1;
INSERT 0 1000000

-- Check the DN memory status and table information. Some new data may exist in the TMU, and the
value of htap_used_memory increases after the CU is rebuilt.
-- After a period
htapdb=# EXECUTE DIRECT ON (dn_6001_6002) $$SELECT * FROM pg_total_memory_detail WHERE
memorytype='max_htap_memory' OR memorytype='htap_used_memory'$$;
   nodename   |     memorytype     | memorymbytes
--------------+--------------------+--------------
 dn_6001_6002 | max_htap_memory  |         1024
 dn_6001_6002 | htap_used_memory |            6
(2 rows)

htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info();
  node_name   |      rel_name       | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem | cu_mem_size |
cu_num_in_disk | cu_disk_size
--------------+---------------------+-----------+---------+--------------+---------------+-------------+----------------
+--------------
 dn_6001_6002 | htap_imcv_mem_ctl_1 |         4 | 1 3 4 5 |            6 |            25 |     6303358 |              0
|            0
 dn_6003_6004 | htap_imcv_mem_ctl_1 |         4 | 1 3 4 5 |            6 |            25 |     6295427 |              0
|            0
(2 rows)

htapdb=# DROP TABLE htap_imcv_mem_ctl_1;
DROP TABLE
```

7.20.2.3 DDL

7.20.2.3.1 Creating and Unloading an IMCV

Creating an IMCV for an Ordinary Table

- Syntax
  - HTAP supports IMCV creation using ALTER for an ordinary table, as shown in **Table 7-25**.

    For details, see "SQL Reference > SQL Syntax > A > ALTER TABLE" in *Reference*.

    Table 7-25 ALTER syntaxes for creating an IMCV and impact on the table

Syntax	Loaded Column
ALTER TABLE ... COLVIEW	All columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax	Loaded Column
ALTER TABLE ... NOCOLVIEW	None. The IMCV information is deleted.
ALTER TABLE ... NOCOLVIEW COLVIEW(col_list)	All columns specified in **col_list**.
ALTER TABLE ... COLVIEW NOCOLVIEW(col_list)	Columns that are not listed in **col_list**.

  - HTAP supports IMCV creation using CREATE for an ordinary table. When creating an ordinary table, you can specify a table-level IMCV or column-level IMCV.

    For details, see "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Reference*.

    Table-level IMCV and column-level IMCV affect each other, determining the final IMCV columns to be loaded. **Table 7-26** describes the relationship between table-level and column-level IMCV keywords.

    Table 7-26 Table-level and column-level IMCV keywords and the final impact

Table-Level IMCV Keyword	Column-Level IMCV Keyword	Final Loaded Column
COLVIEW	None/COLVIEW	All columns.
COLVIEW	None/ COLVIEW/ NOCOLVIEW	Columns without NOCOLVIEW defined.
NOCOLVIEW /None	None/ COLVIEW/ NOCOLVIEW	Columns with COLVIEW defined.

- Examples
  - Create an IMCV using ALTER.

```
-- Create an ordinary table htap_test and insert data into the table.
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) with (storage_type = ustore) distribute by hash(id);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (generate_series(1,100), 1, 8000, 'Allen', 'For test');
INSERT 0 100

-- Run ALTER COLVIEW to create an IMCV.
htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE
htapdb=# SELECT total FROM gs_imcv_flush('htap_test');
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 total
-------
 2
(1 row)

htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |         5 | 1 2 3 4 5 |            1 |             6 |        8153 |              0
|              0
 dn_6003_6004 | htap_test |         5 | 1 2 3 4 5 |            1 |             6 |        8185 |              0
|              0
(2 rows)

-- Unload the IMCV.
htapdb=# ALTER TABLE htap_test NOCOLVIEW;
ALTER TABLE

htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
 node_name | rel_name | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem | cu_mem_size
| cu_num_in_disk | cu_disk_size
-----------+----------+-----------+---------+--------------+---------------+-------------+----------------
+--------------
(0 rows)

-- Load columns except dept_id and comments.
htapdb=# ALTER TABLE htap_test COLVIEW NOCOLVIEW(dept_id, comments);
ALTER TABLE

htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+-----------+---------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |         3 | 1 3 4   |            1 |             4 |        8153 |              0
|              0
 dn_6003_6004 | htap_test |         3 | 1 3 4   |            1 |             4 |        8185 |              0
|              0
(2 rows)

-- Unload the IMCV.
htapdb=# ALTER TABLE htap_test NOCOLVIEW;
ALTER TABLE

-- Load the id and name columns.
htapdb=# ALTER TABLE htap_test NOCOLVIEW COLVIEW(id, name);
ALTER TABLE

htapdb=# SELECT node_name, rel_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem,
cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+-----------+---------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |         2 | 1 4     |            1 |             3 |        8153 |              0
|              0
 dn_6003_6004 | htap_test |         2 | 1 4     |            1 |             3 |        8185 |              0
|              0
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE htap_test;
DROP TABLE
```

NOTE

gs_imcv is a shared system catalog related to HTAP. For details, see "System Catalogs and System Views > System Catalogs > HTAP > GS_IMCV" in *Reference*.

- During table creation, set the table-level IMCV keyword to **COLVIEW** and do not specify the column-level IMCV keyword.

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) COLVIEW with (storage_type = ustore) distribute by hash(id);
CREATE TABLE

-- Query the gs_imcv system catalog. The IMCV is loaded to all columns in the table.
htapdb=# SELECT relname,imcvkey FROM gs_imcv;
  relname  |  imcvkey
-----------+-----------
 htap_test | 1 2 3 4 5
(1 row)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

- During table creation, set the table-level IMCV keyword to **COLVIEW**, set the column-level IMCV keyword to **NOCOLVIEW** for some columns, and do not specify the IMCV keyword for other columns.

```
htapdb=#  CREATE TABLE htap_test(
        id int,
        dept_id int NOCOLVIEW,
        salary int,
        name varchar(20) NOCOLVIEW,
        comments varchar(30)) COLVIEW WITH (storage_type = ustore) DISTRIBUTE BY
HASH(id);
CREATE TABLE

-- Query the gs_imcv system catalog. The IMCV is loaded to columns id, salary, and comments.
htapdb=# SELECT relname,imcvkey FROM gs_imcv;
  relname  | imcvkey
-----------+---------
 htap_test | 1 3 5
(1 row)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

- During table creation, do not specify the table-level IMCV keyword, set the IMCV keyword to **COLVIEW** for some columns, and do not specify the IMCV keyword for other columns.

```
htapdb=# CREATE TABLE htap_test(
        id int COLVIEW,
        dept_id int,
        salary int,
        name varchar(20) COLVIEW,
        comments varchar(30)) WITH (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE

-- Query the gs_imcv system catalog. The IMCV is loaded to columns id and name.
htapdb=# SELECT relname,imcvkey FROM gs_imcv;
  relname  | imcvkey
-----------+---------
 htap_test | 1 4
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE htap_test;
DROP TABLE
```

Creating an IMCV for a Partitioned Table

- Syntax
  - HTAP supports IMCV creation using ALTER for a partitioned table. Partitioned tables inherit the syntax of creating an IMCV using ALTER from ordinary tables. The ALTER syntax for partitions is added for partitioned tables as follows:

    ```
    ALTER TABLE table_name MODIFY PARTITION part_name COLVIEW;
    ```

    For details, see "SQL Reference > SQL Syntax > A > ALTER TABLE" in *Reference*.
  - HTAP supports IMCV creation using CREATE. When creating a partitioned table, you can specify a table-level IMCV, column-level IMCV, or partition-level IMCV.

    For details, see "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Reference*.

    For a partitioned table, the columns loading an IMCV are also determined by the table-level IMCV and column-level IMCV. The rules are the same as those for an ordinary table. Partitioned tables have partition-level IMCVs. The partition-level IMCV and table-level IMCV affect each other and determine which partitions load an IMCV. The IMCV column information of partitions is the same as that of the base table. **Table 7-27** describes the relationship between table-level and partition-level IMCV keywords.

    Table 7-27 Table-level and partition-level IMCV keywords and the final impact

Table-Level IMCV Keyword	Partition-Level IMCV Keyword	IMCV Created for the Partition
COLVIEW	None/ COLVIEW	Yes
None/ COLVIEW/ NOCOLVIEW	NOCOLVIEW	No
None/ NOCOLVIEW	COLVIEW	Yes

- Examples
  - Load an IMCV to the specified partition.

    ```
    -- Create a partitioned table htap_test, and insert data into the table.
    htapdb=# CREATE TABLE htap_test(
            id int,
            dept_id int,
            salary int,
            name varchar(20),
            comments varchar(30)
            ) WITH(storage_type = ustore) DISTRIBUTE BY HASH(id)
    ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        PARTITION BY HASH (id)(
        PARTITION id_p1,
        PARTITION id_p2,
        PARTITION id_p3);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (generate_series(1,100), 1, 8000, 'Allen', 'For test');
INSERT 0 100

-- No IMCV is loaded to the htap_test table, and no record is found in the gs_imcv system
catalog.
htapdb=# SELECT relname,imcvkey FROM gs_imcv;
 relname | imcvkey
---------+---------
(0 rows)

-- Use ALTER to load an IMCV to the specified partition id_p1.
htapdb=# ALTER TABLE htap_test MODIFY PARTITION id_p1 COLVIEW;
ALTER TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info()
ORDER BY node_name, rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num |
cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------
+-------------+----------------+--------------
 dn_6001_6002 | htap_test | id_p1          |         5 | 1 2 3 4 5 |            1 |             6 |        8101
|             0 |              0
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0
|             0 |              0
 dn_6003_6004 | htap_test | id_p1          |         5 | 1 2 3 4 5 |            1 |             6 |        8101
|             0 |              0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0
|             0 |              0
(4 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

- When creating a partitioned table, set the table-level IMCV keyword to **COLVIEW** and do not specify the partition-level IMCV keyword.

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)
        ) COLVIEW WITH(storage_type = ustore) DISTRIBUTE BY HASH(id)
        PARTITION BY HASH (id)(
        PARTITION id_p1,
        PARTITION id_p2,
        PARTITION id_p3);
CREATE TABLE

-- Load an IMCV to all partitions.
htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info()
ORDER BY node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num |
cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------
+-------------+----------------+--------------
 dn_6001_6002 | htap_test | id_p1          |         5 | 1 2 3 4 5 |            0 |             0 |           0
|             0 |              0
 dn_6001_6002 | htap_test | id_p2          |         5 | 1 2 3 4 5 |            0 |             0 |           0
|             0 |              0
 dn_6001_6002 | htap_test | id_p3          |         5 | 1 2 3 4 5 |            0 |             0 |           0
|             0 |              0
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
|           0 |           0
 dn_6003_6004 | htap_test | id_p1          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test | id_p2          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test | id_p3          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test |                |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
(8 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

- When creating a partitioned table, set the table-level IMCV keyword to **COLVIEW** and the partition-level IMCV keyword of some partitions to **NOCOLVIEW**.

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)
        ) COLVIEW WITH(storage_type = ustore) DISTRIBUTE BY HASH(id)
        PARTITION BY HASH (id)(
        PARTITION id_p1,
        PARTITION id_p2 NOCOLVIEW,
        PARTITION id_p3 );
CREATE TABLE

-- Load an IMCV to all partitions except id_p2.
htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info()
ORDER BY node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num |
cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------
+-------------+----------------+--------------
 dn_6001_6002 | htap_test | id_p1          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6001_6002 | htap_test | id_p3          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6001_6002 | htap_test |                |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test | id_p1          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test | id_p3          |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
 dn_6003_6004 | htap_test |                |       5 | 1 2 3 4 5 |          0 |           0 |         0
|           0 |           0
(6 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

- When creating a partitioned table, set the table-level IMCV keyword to **NOCOLVIEW** and the partition-level IMCV keyword of some partitions to **COLVIEW**.

```
-- Partition.
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)
        ) NOCOLVIEW WITH(storage_type = ustore) DISTRIBUTE BY HASH(id)
        PARTITION BY HASH (id)(
        PARTITION id_p1,
        PARTITION id_p2 COLVIEW,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        PARTITION id_p3 );
CREATE TABLE

-- Load an IMCV to the id_p2 partition. Other partitions are not loaded with the IMCV.
htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info()
ORDER BY node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num |
cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------
+-------------+----------------+--------------
 dn_6001_6002 | htap_test | id_p2          |         5 | 1 2 3 4 5 |            0 |            0 |           0
|             0 |            0
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |            0 |           0
|             0 |            0
 dn_6003_6004 | htap_test | id_p2          |         5 | 1 2 3 4 5 |            0 |            0 |           0
|             0 |            0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |            0 |           0
|             0 |            0
(4 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

Unloading an IMCV from an Ordinary Table

- Syntax

  Use ALTER to delete IMCV column-store data and corresponding IMCV metadata.

  ```
  ALTER TABLE table_name NOCOLVIEW;
  ```

- Examples

  Delete an IMCV by running **ALTER TABLE ... NOCOLVIEW**.

  ```
  htapdb=# CREATE TABLE htap_test(
          id int,
          dept_id int,
          salary int,
          name varchar(20),
          comments varchar(30)) COLVIEW WITH (storage_type = ustore) DISTRIBUTE BY HASH(id);
  CREATE TABLE

  htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
  cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
  node_name,rel_name;
    node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem
  | cu_mem_size | cu_num_in_disk | cu_disk_size
  --------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
  +----------------+--------------
   dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |            0 |           0 |            0
  |              0
   dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |            0 |           0 |            0
  |              0
  (2 rows)

  htapdb=# ALTER TABLE htap_test NOCOLVIEW;
  ALTER TABLE

  -- After the IMCV is deleted, no record is found when the gs_global_imcv_info() system function is
  used for query.
  htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
  cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
  node_name,rel_name;
   node_name | rel_name | partition_name | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem |
  cu_mem_size | cu_num_in_disk | cu_disk_size
  -----------+----------+----------------+-----------+---------+--------------+---------------+-------------
  +----------------+--------------
  (0 rows)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE htap_test;
DROP TABLE
```

Unloading an IMCV from a Partitioned Table

- Syntax

  Partitioned tables inherit the syntax of deleting an IMCV from ordinary tables. In addition, you can delete IMCV column-store data and IMCV metadata from specified partitions.

  ```
  ALTER TABLE table_name MODIFY PARTITION part_name NOCOLVIEW;
  ```

- Examples

  Delete an IMCV from a partitioned table.

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)
 ) WITH (storage_type = ustore) DISTRIBUTE BY HASH(id)
 PARTITION BY RANGE (id)
 (
  PARTITION id_p1 VALUES LESS THAN (100),
  PARTITION id_p2 VALUES LESS THAN (200) COLVIEW,
  PARTITION id_p3 VALUES LESS THAN (300) COLVIEW,
  PARTITION id_p4 VALUES LESS THAN (400) ,
  PARTITION id_p5 VALUES LESS THAN (MAXVALUE));
CREATE TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem
| cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test | id_p3          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |             0
|              0
 dn_6001_6002 | htap_test | id_p2          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
 dn_6003_6004 | htap_test | id_p3          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |             0
|              0
 dn_6003_6004 | htap_test | id_p2          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
(6 rows)

htapdb=# ALTER TABLE htap_test MODIFY PARTITION id_p2 NOCOLVIEW;
ALTER TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem
| cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test | id_p3          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |             0
|              0
 dn_6003_6004 | htap_test | id_p3          |         5 | 1 2 3 4 5 |            0 |             0 |           0 |
0 |              0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |             0
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
|           0
(4 rows)

htapdb=# ALTER TABLE htap_test DROP PARTITION id_p3;
ALTER TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
 node_name | rel_name | partition_name | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
-----------+----------+----------------+-----------+---------+--------------+---------------+-------------
+----------------+--------------
(0 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

7.20.2.3.2 Other DDL Statements

TRUNCATE

When a table is truncated, the row-store data is cleared, and the IMCV column-store data loaded in the table is cleared in cascading mode. The IMCV metadata is not deleted.

When a partition is truncated, the row-store partition data is cleared, and the IMCV column-store data in the partition is cleared in cascading mode. The IMCV metadata is not deleted.

Example:

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) WITH (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (generate_series(1,30), 1, 8000, 'Alice', 'for test');
INSERT 0 30

htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE

htapdb=# SELECT count(*) FROM htap_test;
 count
-------
    30
(1 row)

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            1 |             6 |        8099 |              0
|              0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            1 |             6 |        8099 |              0
|              0
(2 rows)

htapdb=# TRUNCATE TABLE htap_test;
TRUNCATE TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# SELECT count(*) FROM htap_test;
 count
-------
     0
(1 row)

-- TRUNCATE does not delete the IMCV metadata. The data records can still be queried using the system function gs_global_imcv_info().
htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------+----------------+--------------
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |              0 |            0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |              0 |            0
(2 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

DROP

When a table is dropped, the row-store table data is dropped, and all IMCV column-store data and corresponding IMCV metadata in the table are dropped in cascading mode.

When a partition is dropped, the partition data in the row-store table is dropped, and all IMCV column-store data and corresponding IMCV metadata in the partition are dropped in cascading mode.

Example:

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) with (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (generate_series(1,30), 1, 8000, 'Alice', 'for test');
INSERT 0 30

htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE

htapdb=# SELECT count(*) FROM htap_test;
 count
-------
    30
(1 row)

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num, cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem | cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------+----------------+--------------
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |              0 |            0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            0 |             0 |           0 |              0 |            0
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE htap_test;
DROP TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
 node_name | rel_name | partition_name | imcvnattr | imcvkey | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
-----------+----------+----------------+-----------+---------+--------------+---------------+-------------
+----------------+--------------
(0 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

VACUUM FULL

When VACUUM FULL is performed on row-store tables, it takes effect on column-store data in cascading mode.

The processing of HTAP column-store data consists of two phases:

1. Delete the original IMCV information.
2. Rebuild the IMCV and reloading IMCV column-store data to achieve memory compactness.

Example:

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) WITH (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (generate_series(1,30), 1, 8000, 'Alice', 'for test');
INSERT 0 30

htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |                |         5 | 1 2 3 4 5 |            1 |             6 |        8099 |              0
|              0
 dn_6003_6004 | htap_test |                |         5 | 1 2 3 4 5 |            1 |             6 |        8099 |              0
|              0
(2 rows)

htapdb=# DELETE FROM htap_test WHERE id<10;
DELETE 9

htapdb=# VACUUM FULL htap_test;
VACUUM

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   | rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
--------------+-----------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | htap_test |              |         5 | 1 2 3 4 5 |           1 |            6 |       8087 |             0
|              0
 dn_6003_6004 | htap_test |              |         5 | 1 2 3 4 5 |           1 |            6 |       8093 |             0
|              0
(2 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

7.20.2.4 DML/DQL

7.20.2.4.1 DML

In HTAP, the TMU records incremental data changes caused by DML operations and maintains data freshness during IMCV column query, ensuring data consistency with real-time column-store and row-store data.

Example:

```
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) with (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO htap_test VALUES (1, 1, 2000, 'Allen', 'for test');
INSERT 0 1

htapdb=# INSERT INTO htap_test VALUES (2, 2, 3000, 'Lily', 'for test');
INSERT 0 1

htapdb=# INSERT INTO htap_test VALUES (3, 2, 4000, 'Ann', 'for test');
INSERT 0 1

-- Load an IMCV.
htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE

htapdb=# SELECT reloid, relname, dbname, username, parentoid, imcvispart, imcvnattr, imcvkey, priority
FROM gs_imcv;
 reloid |  relname  | dbname | username | parentoid | imcvispart | imcvnattr |  imcvkey  | priority
--------+-----------+--------+----------+-----------+------------+-----------+-----------+----------
  17949 | htap_test | test   | dbdev    |         0 | f          |         5 | 1 2 3 4 5 |        1
(1 row)

-- Insert new data into the table.
htapdb=# INSERT INTO htap_test VALUES (4, 2, 3000, 'Bob', 'for test');
INSERT 0 1

htapdb=# SET enable_stream_operator=on;
SET

htapdb=# SET enable_fast_query_shipping=off;
SET

-- Generate an IMCV execution plan and execute it.
htapdb=# EXPLAIN SELECT /*+imcvscan(htap_test)*/  * FROM htap_test;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Row Adapter  (cost=36.74..36.74 rows=100 width=27)
   ->  Vector Streaming (type: GATHER)  (cost=3.12..36.74 rows=100 width=27)
         Node/s: All datanodes
         ->  Imcv Scan on htap_test  (cost=0.00..32.05 rows=100 width=27)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(4 rows)

-- Obtain the IMCV query result, which is consistent with the row-store data.
htapdb=# SELECT /*+imcvscan(htap_test)*/  * FROM htap_test;
 id | dept_id | salary | name  | comments
----+---------+--------+-------+----------
  1 |       1 |   2000 | Allen | for test
  2 |       2 |   3000 | Lily  | for test
  3 |       2 |   4000 | Ann   | for test
  4 |       2 |   3000 | Bob   | for test
(4 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

7.20.2.4.2 DQL

After correctly setting parameters and creating an IMCV for a table, you can perform DQL queries on the table. When **htap_router_mode** is set to **auto**, the transparent router for HTAP selects the optimal execution plan based on data and SQL characteristics. Therefore, even if the IMCV is loaded to the target table, the transparent router may still select a row-store plan based on the cost calculation result. You can run the **EXPLAIN** command to view the execution plan.

If you want to use an IMCV column-store plan during SQL execution, set **htap_router_mode** to **column** or set an IMCV hint in SQL statements.

For details about how to use transparent router, see **Transparent Routing for HTAP**.

Example:

```
htapdb=# SET htap_router_mode='auto';
SET
htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 auto
(1 row)

-- Create a table, insert data, and load an IMCV.
htapdb=# CREATE TABLE htap_test(
        id int,
        dept_id int,
        salary int,
        name varchar(20),
        comments varchar(30)) WITH (storage_type = ustore) DISTRIBUTE BY HASH(id);
CREATE TABLE
htapdb=# INSERT INTO htap_test VALUES (generate_series(1,100), 1, 8000, 'Allen', 'For test');
INSERT 0 100
htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE
htapdb=# SELECT * FROM gs_imcv;
 reloid |  relname  | dbname | username | parentoid | imcvispart | imcvnattr |  imcvkey  | priority
--------+-----------+--------+----------+-----------+------------+-----------+-----------+----------
  16982 | htap_test | test   | dbdev    |         0 | f          |         5 | 1 2 3 4 5 |        1
(1 row)

-- Because the HTAP routing mode is auto, the transparent router selects a row-store execution plan based on the cost calculation result.
htapdb=# EXPLAIN SELECT count(*) FROM htap_test;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=13.23..13.27 rows=1 width=8)
   ->  Streaming (type: GATHER)  (cost=13.23..13.27 rows=2 width=8)
         Node/s: All datanodes
         ->  Aggregate  (cost=13.16..13.17 rows=2 width=8)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
            ->  Seq Scan on htap_test  (cost=0.00..13.13 rows=20 width=0)
(5 rows)

-- Use an IMCV hint to specify an IMCV execution plan.
htapdb=# EXPLAIN SELECT /*+imcvscan(htap_test)*/  count(*) FROM htap_test;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Row Adapter  (cost=32.28..32.28 rows=1 width=8)
   ->  Vector Aggregate  (cost=32.24..32.28 rows=1 width=8)
         ->  Vector Streaming (type: GATHER)  (cost=32.24..32.28 rows=2 width=8)
               Node/s: All datanodes
               ->  Vector Aggregate  (cost=32.17..32.18 rows=2 width=8)
                     ->  Imcv Scan on htap_test  (cost=0.00..32.05 rows=100 width=0)
(6 rows)

htapdb=# SET htap_router_mode='column';
SET
htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 column
(1 row)

-- Forcibly choose an IMCV execution plan when the HTAP routing mode is set to column.
htapdb=# EXPLAIN SELECT count(*) FROM htap_test;
                             QUERY PLAN
---------------------------------------------------------------------------
 Aggregate  (cost=2.69..2.73 rows=1 width=8)
   ->  Streaming (type: GATHER)  (cost=2.69..2.73 rows=2 width=8)
         Node/s: All datanodes
         ->  Aggregate  (cost=2.62..2.63 rows=2 width=8)
               ->  Seq Scan on htap_test  (cost=0.00..2.50 rows=100 width=0)
(5 rows)

htapdb=# DROP TABLE htap_test;
DROP TABLE
```

7.20.2.4.3 Column-to-Row Hybrid Scan

Column-to-row hybrid scan is a special column-store scan mode. If not all columns accessed by a query are created as column-store and all columns involved in the query predicate are created as IMCV, a column-to-row hybrid scan plan can be generated based on the cost.

When the memory is limited and an IMCV is created only for some columns, this feature can be used to accelerate the query by using these columns.

Example:

```
-- If only some columns have column-store data, the column-to-row hybrid scan can be used.
htapdb=# DROP TABLE IF EXISTS test_table;
CREATNOTICE:  table "test_table" does not exist, skipping
DROP TABLE

htapdb=# CREATE TABLE test_table (
    supplier_txn_base_id bigint NOT NULL,
    supplier_id bigint NOT NULL,
    supplier_address_id bigint,
    supplier_txn_code nvarchar2(30) NOT NULL,
    shipment_code nvarchar2(30),
    invoice_currency_code nvarchar2(15)) DISTRIBUTE BY HASH(supplier_txn_base_id);
CREATE TABLE

htapdb=# INSERT INTO test_table VALUES (200, 100, -1, 'CNY', 'ee', 'CNY');
INSERT 0 1

htapdb=# INSERT INTO test_table VALUES (300, 200, -1, 'EUR', 'de', 'EUR');
INSERT 0 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# INSERT INTO test_table VALUES (200, 300, -1, 'US', 'de', 'US');
INSERT 0 1

htapdb=# SET enable_imcvscan=on;
SET

htapdb=# SET htap_router_mode='column';
SET

-- Create an IMCV for some columns.
htapdb=# ALTER TABLE test_table COLVIEW(supplier_id, supplier_txn_code);
ALTER TABLE

htapdb=# SELECT node_name, rel_name, partition_name, imcvnattr, imcvkey, rowgroup_num,
cu_num_in_mem, cu_mem_size, cu_num_in_disk, cu_disk_size FROM gs_global_imcv_info() ORDER BY
node_name,rel_name;
  node_name   |  rel_name  | partition_name | imcvnattr |  imcvkey  | rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
--------------+------------+----------------+-----------+-----------+--------------+---------------+-------------
+----------------+--------------
 dn_6001_6002 | test_table |                |         2 | 2 4       |            1 |             3 |       12112 |              0
|              0
 dn_6003_6004 | test_table |                |         2 | 2 4       |            0 |             0 |           0 |              0
|              0
(4 rows)

-- Disable column-to-row hybrid scan.
htapdb=# SET enable_hybrid_imcvscan=off;
SET

htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_table t WHERE t.supplier_id >= 200;
              QUERY PLAN
---------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_table t
         Filter: (supplier_id >= 200)
(4 rows)
-- Enable column-to-row hybrid scan.
htapdb=# SET enable_hybrid_imcvscan=on;
SET
-- The query result shows the columns without an IMCV created and the predicate columns with an IMCV
created. In this case, column-store scan is still supported.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_table t WHERE t.supplier_id >= 200;
                 QUERY PLAN
--------------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_table t
               Filter: (supplier_id >= 200)
(5 rows)

htapdb=# DROP TABLE test_table;
DROP TABLE
```

7.20.2.5 Transparent Routing for HTAP

HTAP supports multiple query routing modes: automatic routing, forcible column store, and forcible row store. You can set **htap_router_mode** to **'auto'**, **'column'**, or **'row'** respectively to select a routing mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Preparation

Create a small table **test_small** with several rows of data and a large table **test_large** with 3000 rows of data, and insert data into the tables. Row-store scan is faster for the small table, and column-store scan is faster for the large table.

Example:

```
htapdb=# CREATE TABLE test_small (
                        id int,
                        dept_id int,
                        salary int,
                        name varchar(20),
                        comments varchar(30)) DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO test_small VALUES (1, 1, 1000, 'Jack', 'AAAA');
INSERT 0 1

htapdb=# INSERT INTO test_small VALUES (2, 4, 2000, 'Sam', 'BBBB');
INSERT 0 1

htapdb=# INSERT INTO test_small VALUES (3, 2, 3000, 'Sara', 'CCCC');
INSERT 0 1

htapdb=# INSERT INTO test_small VALUES (4, 6, 4000, 'Oulu', 'DDDD');
INSERT 0 1

htapdb=# INSERT INTO test_small VALUES (5, 3, 5000, 'John', 'EEEE');
INSERT 0 1

htapdb=# CREATE TABLE test_large (
                        id int,
                        dept_id int,
                        salary int,
                        name varchar(20),
                        comments varchar(30))DISTRIBUTE BY HASH(id);
CREATE TABLE

htapdb=# INSERT INTO test_large VALUES (generate_series(1,15000), 1, 1000, 'large', 'large table');
INSERT 0 15000

-- Create an IMCV for the entire test_small table.
htapdb=# ALTER TABLE test_small COLVIEW;
ALTER TABLE

-- Create an IMCV for some columns in the test_large table.
htapdb=# ALTER TABLE test_large COLVIEW(id,dept_id);
ALTER TABLE
```

Automatic Routing Mode

You can set **htap_router_mode** to **'auto'** to enable the automatic routing mode. In automatic routing mode, a row-based, column-based, or row-column hybrid execution plan is automatically selected based on whether an IMCV is created for related tables and columns and the cost of a row-column execution plan.

Example:

```
htapdb=# SET htap_router_mode = 'auto';
SET

htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 auto
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(1 row)

-- The cost of a row-store plan is lower. Therefore, a row-store plan is selected.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_small;
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_small
(3 rows)

-- The cost of a column-store plan is lower. Therefore, a column-store plan is selected.
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM test_large;
             QUERY PLAN
-------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_large
(4 rows)

-- No IMCV is created for the target columns. Therefore, a row-store plan is selected.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_large;
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_small
(3 rows)
```

Forcible Column Store Mode

If AP loads dominate, you can set **htap_router_mode** to **'column'** to enable the forcible column store mode. In this mode, when IMCVs exist in the relevant tables and columns queried, if the query result shows that conditions for column store are met, a column-store execution plan is preferentially selected regardless of the cost.

Example:

```
htapdb=# SET htap_router_mode = 'column';
SET
htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 column
(1 row)

-- An IMCV has been created for all columns in the table. Therefore, the column-store plan is executed.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_small;
             QUERY PLAN
-------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_small
(4 rows)

-- An IMCV has been created for the target columns. Therefore, the column-store plan is executed.
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM test_large;
             QUERY PLAN
-------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_large
(4 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- No IMCV has been created for the target columns. Therefore, the row-store plan is executed.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_large;
          QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_small
(3 rows)
```

Forcible Row Store Mode

If TP loads dominate, set **htap_router_mode** to **'row'** to enable the forcible row store mode. In this mode, the row-store plan is selected regardless of whether the conditions for a column-store execution plan are met.

Example:

```
htapdb=# SET htap_router_mode = 'row';
SET
htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 row
(1 row)

-- Forcibly execute a row-store plan.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_small;
          QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_small
(3 rows)

-- Forcibly execute a row-store plan.
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM test_large;
          QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_large
(3 rows)

-- Forcibly execute a row-store plan.
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM test_large;
          QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_large
(3 rows)
```

Hint-specified Routing Mode

You can use hints to specify the column-store scan execution plan of a specified table for a single query. That is, when the specified table has column-store data and the loaded columns meet the query requirements, the table is forcibly scanned in column store mode. For example, the functionality can be implemented based on the hint syntax /* + imcvscan (tbl) */.

Example:

```
htapdb=# SET htap_router_mode = 'row';
SET
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# SHOW htap_router_mode;
 htap_router_mode
------------------
 row
(1 row)

-- Although a row-store plan costs less, a column-store plan is still selected based on a hint.
htapdb=# EXPLAIN (COSTS OFF) SELECT /*+imcvscan (test_small)*/ * FROM test_small;
             QUERY PLAN
---------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_small
(4 rows)

-- Forcibly select a column-store plan based on a hint.
htapdb=# EXPLAIN (COSTS OFF) SELECT /*+imcvscan (test_large)*/ id,dept_id FROM test_large;
             QUERY PLAN
---------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on test_large
(4 rows)

-- Accelerated columns in the table specified by a hint do not meet the query requirements. Therefore, a
row-store plan is selected.
htapdb=# EXPLAIN (COSTS OFF) SELECT /*+imcvscan (test_large)*/ * FROM test_large;
          QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on test_large
(3 rows)

-- Perform column-store scan on the test_large table specified by a hint.
htapdb=# EXPLAIN (COSTS OFF) SELECT /*+imcvscan (test_large)*/ test_small.id, test_large.dept_id FROM
test_small, test_large WHERE test_small.id = test_large.id;
                     QUERY PLAN
------------------------------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Vector Ca Hash Join
               Hash Cond: (test_large.id = test_small.id)
               ->  Imcv Scan on test_large
               ->  Vector Adapter(type: BATCH MODE)
                     ->  Seq Scan on test_small
(8 rows)

-- Perform column-store scan on the two tables joined by using a hint.
htapdb=# EXPLAIN (COSTS OFF) SELECT /*+imcvscan (test_large) imcvscan (test_small)*/ test_small.id,
test_large.dept_id FROM test_small, test_large WHERE test_small.id = test_large.id;
                     QUERY PLAN
------------------------------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Vector Ca Hash Join
               Hash Cond: (test_small.id = test_large.id)
               ->  Imcv Scan on test_small
               ->  Imcv Scan on test_large
(7 rows)

htapdb=# DROP TABLE test_large;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE test_small;
DROP TABLE
```

7.20.2.6 Column-Store Data Maintenance

Column-Store Data Loading

Column-store data loading refers to the process of generating column-store data for a table based on row-store data for the first time. After the asynchronous loading mode is enabled, when you run a DDL command to create an IMCV for a table, the system only creates the column-store metadata structure and does not immediately generate the actual column-store data.

The system provides two methods for loading column-store data:

- Automatic loading by the background HTAP workers

  The system schedules HTAP workers in the background to automatically build column-store data for the target table. This method does not require manual intervention. However, the execution time may be affected by the number of background HTAP worker threads (determined by the GUC parameter **htap_flush_workers**), system load, and scheduling policy.

- Manual triggering of loading

  You can directly call the gs_imcv_flush() API to immediately load column-store data to a specified table. If the background loading is not completed in time, you can proactively execute this API to accelerate the loading process. The loading efficiency is related to the degree of parallelism specified by the parameter.

NOTE

- The number of HTAP workers is determined by the GUC parameter **htap_flush_workers**. For details about how to set this parameter, see "Database GUC Parameters > GUC Parameters > Hybrid Transactional/Analytical Processing" in *Reference*.
- gs_imcv_flush is a system function used to load or rebuild column-store data. For details about how to use this function, see "SQL Reference > Functions and Operators > System Administration Functions > HTAP System Functions" in *Reference*.

Example of automatic loading:

```
-- Create an ordinary table and insert data into the table.
htapdb=# DROP TABLE IF EXISTS htap_gs_imcv_flush_nopart;
htapdb=# CREATE TABLE htap_gs_imcv_flush_nopart(
    c1 bigint,
    c2 varchar(256)
) WITH (STORAGE_TYPE=USTORE) DISTRIBUTE BY HASH(c1);
CREATE TABLE

htapdb=# INSERT INTO htap_gs_imcv_flush_nopart (c1, c2)
SELECT
    g AS c1,
    'data_' || g AS c2
FROM generate_series(1, 256000) AS g;
INSERT 0 256000

-- Create a partitioned table and insert data into it.
htapdb=# DROP TABLE IF EXISTS htap_gs_imcv_flush_part;
htapdb=# CREATE TABLE htap_gs_imcv_flush_part (
    c1 bigint,
    c2 bigint,
    c3 varchar(256)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
) DISTRIBUTE BY HASH(c1)
PARTITION BY RANGE (c1)(
    PARTITION htap_gs_imcv_flush_part_p1 VALUES LESS THAN (2),
    PARTITION htap_gs_imcv_flush_part_p2 VALUES LESS THAN (3)
);
CREATE TABLE

htapdb=# INSERT INTO htap_gs_imcv_flush_part (c1, c2, c3)
SELECT
    CASE
        WHEN g <= 256000 THEN 1
        ELSE 2
    END AS c1,
    g AS c2,
    'data_' || g AS c3
FROM generate_series(1, 512000) AS g;

-- Create IMCVs for a non-partitioned table and a partitioned table.
htapdb=# ALTER TABLE htap_gs_imcv_flush_nopart COLVIEW;
ALTER TABLE
htapdb=# ALTER TABLE htap_gs_imcv_flush_part COLVIEW;
ALTER TABLE

-- View the IMCV information on the DN. (In the information, rowgroup_num indicates the actual number of row groups in the table, and populated_rowgroup_num indicates the number of row groups that have been loaded to the table.)
htapdb=# SELECT rel_name, partition_name, rowgroup_num, populated_rowgroup_num FROM gs_htap_imcv_info();
          rel_name          |       partition_name       | rowgroup_num | populated_rowgroup_num
----------------------------+----------------------------+--------------+------------------------
 htap_gs_imcv_flush_part    | htap_gs_imcv_flush_part_p1 |            2 |                      0
 htap_gs_imcv_flush_part    |                            |            0 |                      0
 htap_gs_imcv_flush_part    | htap_gs_imcv_flush_part_p2 |            2 |                      0
 htap_gs_imcv_flush_nopart  |                            |            2 |                      2

-- Wait for a while and check the IMCV information (after the background loading is complete).
htapdb=# SELECT rel_name, partition_name, rowgroup_num, populated_rowgroup_num FROM gs_htap_imcv_info();
          rel_name          |       partition_name       | rowgroup_num | populated_rowgroup_num
----------------------------+----------------------------+--------------+------------------------
 htap_gs_imcv_flush_part    | htap_gs_imcv_flush_part_p1 |            2 |                      2
 htap_gs_imcv_flush_part    |                            |            0 |                      0
 htap_gs_imcv_flush_part    | htap_gs_imcv_flush_part_p2 |            2 |                      2
 htap_gs_imcv_flush_nopart  |                            |            2 |                      2
```

Example of manual loading:

```
-- Create the non-partitioned table htap_gs_imcv_flush_nopart and partitioned table htap_gs_imcv_flush_part again and insert data into them. The creation method is the same as that in the preceding example.
-- 1. Specify the relationship to load column-store data.
-- (1) Specify a non-partitioned table to load column-store data.
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_nopart');
 total | success | error | skip
-------+---------+-------+------
 2     | 2       | 0     | 0
(1 row)
-- (2) Specify a partitioned table to load column-store data.
-- (2.1) Specify a partitioned table to load column-store data.
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', 'htap_gs_imcv_flush_part_p1');
 total | success | error | skip
-------+---------+-------+------
 2     | 2       | 0     | 0
(1 row)
-- (2.2) Specify a base table to load column-store data. (p2: The loading is successful; p1: Loading has been performed, so it is skipped.)
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', '');
 total | success | error | skip
-------+---------+-------+------
 4     | 2       | 0     | 2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(1 row)
-- 2. Specify the degree of parallelism for loading column-store data.
htapdb=# ALTER TABLE htap_gs_imcv_flush_part NOCOLVIEW;
ALTER TABLE
htapdb=# ALTER TABLE htap_gs_imcv_flush_part COLVIEW;
ALTER TABLE
htapdb=# \timing on
Timing is on.
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', '', 2);
 total | success | error | skip
-------+---------+-------+------
 4     | 4       | 0     | 0
(1 row)

Time: 406.391 ms
htapdb=# \time off
Timing is off.
htapdb=# ALTER TABLE htap_gs_imcv_flush_part NOCOLVIEW;
ALTER TABLE
htapdb=# ALTER TABLE htap_gs_imcv_flush_part COLVIEW;
ALTER TABLE
htapdb=# \timing on
Timing is on.
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', '', 2);
 total | success | error | skip
-------+---------+-------+------
 4     | 4       | 0     | 0
(1 row)

Time: 332.194 ms
htapdb=# \time off
Timing is off.
```

Column-Store Data Rebuild

After column-store data is loaded, all subsequent DML (INSERT, UPDATE, and DELETE) operations will generate new incremental data. The incremental data is maintained by the real-time transaction monitoring unit (TMU), but are not synchronized to the loaded column-store data in real time.

When a large amount of incremental data is accumulated, the column-store data gradually lags behind the actual row-store content, affecting the column-store scan performance. In this case, you need to rebuild column-store data and synchronize the accumulated incremental data to the column-store data to ensure the freshness and query performance of the column-store data.

The system also provides two rebuild methods:

- Automatic rebuild by the background HTAP workers

  When the incremental data reaches a certain scale or a certain period of time has passed since the last rebuild, the system automatically triggers a column-store rebuild task in the background. The rebuild duration is also affected by the number of background threads and thread scheduling.

  NOTE

  The conditions for triggering automatic rebuild in the background are as follows:

  - The number of pages involved in incremental data reaches a certain proportion, which is controlled by the GUC parameter **htap_flush_pagenum**.
  - The interval since the last rebuild reaches a certain value, which is controlled by the GUC parameter **htap_flush_interval**.

  For details, see "Database GUC Parameters > GUC Parameters > Hybrid Transactional/Analytical Processing" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Manual rebuild

  If the background rebuild is not performed promptly or you want to update the column-store content immediately, you can directly call the gs_imcv_flush() API to manually trigger the rebuild. The rebuild efficiency is related to the degree of parallelism specified by the parameter.

Example of automatic rebuild:

```
-- View the synchronization interval and dirty page threshold.
htapdb=# SHOW htap_flush_interval;
 htap_flush_interval
---------------------
 5min
(1 row)
htapdb=# SHOW htap_flush_pagenum;
 htap_flush_pagenum
--------------------
 512
(1 row)
-- Update data to the partitioned table htap_gs_imcv_flush_part.
htapdb=# UPDATE htap_gs_imcv_flush_part SET c3 = 'abcd' WHERE c2 IN (2, 20002);
UPDATE 2
-- View the number of dirty pages (threshold for the number of the dirty pages that do not meet the rebuild conditions).
htapdb=# SELECT rel_name, sum(dirty_page_count) as dirty_page_count FROM gs_global_htap_tmu_chunk_meta('htap_gs_imcv_flush_part','',0) GROUP BY rel_name;
          rel_name       | dirty_page_count
-------------------------+------------------
 htap_gs_imcv_flush_part |                2
(1 row)
-- Wait for 5 minutes and check the number of dirty pages. At this time, the background rebuild is complete (the background rebuild is triggered when the specified rebuild interval is reached).
htapdb=# SELECT rel_name, sum(dirty_page_count) as dirty_page_count FROM gs_global_htap_tmu_chunk_meta('htap_gs_imcv_flush_part','',0) GROUP BY rel_name;
          rel_name       | dirty_page_count
-------------------------+------------------
 htap_gs_imcv_flush_part |                0
(1 row)
-- Insert sufficient data (greater than the dirty page threshold) into the partitioned table htap_gs_imcv_flush_part.
htapdb=# INSERT INTO htap_gs_imcv_flush_part (c1, c2, c3)
SELECT 1, 1, 'c3' FROM generate_series(1, 120000);
INSERT 120000
-- View the number of dirty pages.
htapdb=# SELECT rel_name, sum(dirty_page_count) as dirty_page_count FROM gs_htap_tmu_chunk_meta('htap_gs_imcv_flush_part','',0) GROUP BY rel_name;
          rel_name       | dirty_page_count
-------------------------+------------------
 htap_gs_imcv_flush_part |              557
(1 row)
-- After the background automatic rebuild is complete, view the number of dirty pages again.
htapdb=# SELECT rel_name, sum(dirty_page_count) as dirty_page_count FROM gs_htap_tmu_chunk_meta('htap_gs_imcv_flush_part','',0) GROUP BY rel_name;
          rel_name       | dirty_page_count
-------------------------+------------------
 htap_gs_imcv_flush_part |                0
(1 row)
```

Example of manual rebuild:

```
-- The method of manually triggering rebuild using the gs_imcv_flush() API is the same as that of manually triggering loading.
-- Insert data into the partitioned table htap_gs_imcv_flush_part.
htapdb=# UPDATE htap_gs_imcv_flush_part SET c3 = 'efgh' WHERE c1=1;
UPDATE 24
-- Manually trigger non-forcible full table rebuild. (If the fourth parameter is missing or set to 0, only the row groups with dirty page marks are rebuilt.)
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', '', 2);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 total | success | error | skip
-------+---------+-------+------
 4    | 1       | 0     | 3
-- Manually trigger forcible full table rebuild. (If the fourth parameter is set to 1, all row groups are rebuilt based on row store.)
htapdb=# SELECT * FROM gs_imcv_flush ('htap_gs_imcv_flush_part', '', 2,1);
 total | success  | error | skip
-------+----------+-------+------
 4    | 4      | 0     | 0
(1 row)

htapdb=# DROP TABLE htap_gs_imcv_flush_nopart;
DROP TABLE

htapdb=# DROP TABLE htap_gs_imcv_flush_part;
DROP TABLE
```

7.20.2.7 Standby Node Capabilities

HTAP does not support standby nodes in distributed mode.

After the **htap_deploy_mode** parameter is set, the value **'primary'** takes effect. That is, HTAP can be enabled only on the primary node.

7.20.2.8 Column-Store Data Disk Extension

HTAP provides the column-store data disk extension functionality. That is, when the actual column-store data space usage is greater than the upper limit of the HTAP memory, the excess part is managed by the disk space. This functionality can be enabled by setting the **htap_memctl_policy** parameter to **'swap'**.

This functionality is used together with hot and cold data management to minimize the disk I/O overhead for accessing data. Column-store metadata is always stored in the memory. When the available HTAP memory is insufficient, the background persists the infrequently accessed column-store data to the disk. When the memory is sufficient, the background loads the hot data from the disk to the memory as much as possible. During column-store scan and query, hot and cold data is not swapped in or out. Instead, the background thread swaps in or out hot and cold data based on the data access frequency.

Example:

```
-- View GUC parameters.
htapdb=# SHOW htap_memctl_policy;
 htap_memctl_policy
--------------------
 swap
(1 row)

htapdb=# SHOW htap_max_mem_size;
 htap_max_mem_size
-------------------
 100MB
(1 row)

htapdb=# CREATE TABLE t1(c1 int, c2 int, c3 varchar(512)) WITH (storage_type=ustore);
CREATE TABLE
htapdb=# INSERT INTO t1 VALUES(
    generate_series(0,2000000),
    generate_series(0,2000000),
    repeat('abcabcabc', 50)
);
INSERT 0 2000001
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# ALTER TABLE t1 COLVIEW;
ALTER TABLE

htapdb=# SELECT 1 FROM gs_imcv_flush('t1');
 ?column?
----------
        1
(1 row)

htapdb=# \x on
Expanded display is on.
-- View the status of the disk files where overflows occur.
htapdb=# SELECT * FROM gs_global_htap_file_status('htapdb');
-[ RECORD 1 ]----+-------------------------
node_name        | dn_6003_6004
db_oid           | 42055
file_name        | global/imcu/42055/2_imcu
file_id          | 2
extent_size      | 1
allocated_blocks | 422
total_blocks     | 16384
high_water_mark  | 1043
utilization      | .0257568
ext              |
-[ RECORD 2 ]----+-------------------------
node_name        | dn_6001_6002
db_oid           | 42085
file_name        | global/imcu/42085/2_imcu
file_id          | 2
extent_size      | 1
allocated_blocks | 701
total_blocks     | 16384
high_water_mark  | 922
utilization      | .0427856
ext              |

htapdb=# DROP TABLE t1;
DROP TABLE
```

7.20.2.9 Column-Store Data Compression

HTAP provides the data compression capability to improve memory resource usage. This functionality can be enabled by setting the **htap_compression_methods** parameter.

Column-store scan requires decompression of compressed data. Therefore, after the compression functionality is enabled, the column-store scan performance is affected to some extent.

A more efficient compression algorithm can be adaptively selected based on the data types. The supported algorithms are Dict, Delta+RLE, and LZ4.

Example:

```
-- View GUC parameters.
htapdb=# SHOW htap_compression_methods;
 htap_compression_methods
--------------------------
 all
(1 row)

-- Create a row-store table and insert data.
htapdb=# CREATE TABLE test_compression(id int, a numeric(15,2), b varchar(100));
CREATE TABLE
htapdb=# INSERT INTO test_compression SELECT id,FLOOR(id/100)+0.5, SUBSTRING(md5(id),0.5) FROM
GENERATE_SERIES(1,1000) AS t(id);
INSERT 0 1000
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create an IMCV.
htapdb=# ALTER TABLE test_compression COLVIEW;
ALTER TABLE

-- Query the compression algorithm and compression ratio of each column in the test_compression table
using the gs_imcu_meta system function.
htapdb=# SELECT node_name, rel_name,imcu_size, imcu_data_size, compressed_data_size,
compressed_ratio FROM gs_global_imcu_meta('test_compression', NULL);
 node_name |     rel_name     | imcu_size | imcu_data_size | compressed_data_size | compressed_ratio

-----------+------------------+-----------+----------------+----------------------+------------------
 datanode1 | test_compression |      5075 |           8008 |                 1075 |           7.4493
 datanode1 | test_compression |      6034 |           4008 |                 2034 |           1.9705
 datanode1 | test_compression |      4083 |          14008 |                   83 |          168.771
 datanode1 | test_compression |     36048 |          36008 |                32048 |          1.12356
(4 rows)

htapdb=# DROP TABLE test_compression;
DROP TABLE
```

7.21 Online DDL Operations

Online DDL operations have little impact on running services, improving service continuity and database availability. Some online DDL operations only modify metadata and do not involve data rebuild. These operations take a short time and have little impact on services. If traditional DDL operations involve data rebuild, such as modifying column types, moving tablespaces, and VACUUM FULL, an exclusive lock will be added to the table for a long time, blocking concurrent read and write operations on the table and affecting user service running. The blocking period is in direct proportion to the data volume. When the data volume is large, the blocking period can be several hours or even days. In contrast, no exclusive lock is added for online DDL operations and concurrent read and write operations can be performed on the table during the execution of online DDL operations, reducing the impact on user services.

NOTE

- Before using the DDL function, ensure that the database is running properly.

Concurrent Read and Write During Online DDL Operations

Step 1 Create a table.

```
gaussdb=# CREATE TABLE t1(c1 INT,c2 INT);
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,10), generate_series(1,10));

-- View the table.
gaussdb=# \d+ t1
                    Table "public.t1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Has OIDs: no
Distribute By: HASH(c1)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off
```

Step 2 Perform online DDL operations that have been supported in earlier versions. (Only some examples are provided. For details about all online DDL operations

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

supported in earlier versions, see "SQL Reference > SQL Syntax > Online DDL and Related Operations" in *Reference*.)

- Add a column online.
```
gaussdb=# ALTER TABLE t1 ADD COLUMN c3 integer;
ALTER TABLE
```
- Delete a column online.
```
gaussdb=# ALTER TABLE t1 drop COLUMN c3;
ALTER TABLE
```
- Rename a table online.
```
gaussdb=# ALTER TABLE t1 RENAME TO t2;
ALTER TABLE
gaussdb=# ALTER TABLE t2 RENAME TO t1;
ALTER TABLE
```

Step 3 Set the keyword **ONLINE** and perform related operations.

- Change the column type online.
```
gaussdb=# ALTER TABLE ONLINE t1 ALTER c2 TYPE BIGINT;
ALTER TABLE
```
- Add a NOT NULL constraint online.
```
gaussdb=# ALTER TABLE ONLINE t1 ALTER c1 SET NOT NULL;
ALTER TABLE
```
- Add a CHECK constraint online.
```
gaussdb=# ALTER TABLE ONLINE t1 ADD CONSTRAINT c2_k CHECK(c2 > 0);
ALTER TABLE
```
- Add and modify a UNIQUE constraint online.
```
gaussdb=# ALTER TABLE ONLINE t1 ADD UNIQUE(c1);
NOTICE:  ALTER TABLE / ADD UNIQUE will create implicit index "t1_c1_key" for table "t1"
ALTER TABLE
```
- Move tablespaces online at the table level.
```
gaussdb=# CREATE TABLESPACE tbs RELATIVE LOCATION 'tablespace1/tbs';
CREATE TABLESPACE
gaussdb=# ALTER TABLE ONLINE t1 SET TABLESPACE tbs;
ALTER TABLE
```
- In the preceding online DDL operations, concurrent read and write operations can be performed.
```
gaussdb=# INSERT INTO t1 VALUES(11, 11);
INSERT 0 1
```

Step 4 View the table definition.

```
gaussdb=# \d+ t1
                        Table "public.t1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer | not null  | plain   |              |
 c2     | bigint  |           | plain   |              |
Indexes:
    "t1_c1_key" UNIQUE CONSTRAINT, ubtree (c1) WITH (storage_type=USTORE) TABLESPACE pg_default
Check constraints:
    "c2_k" CHECK (c2 > 0)
Has OIDs: no
Tablespace: "tbs"
Distribute By: HASH(c1)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, storage_type=USTORE, segment=off
```

Step 5 Run **VACUUM FULL** to recycle tablespaces online.

```
gaussdb=# VACUUM FULL t1 ONLINE;
VACUUM
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Concurrent Partition Read and Write During Online DDL Operations

Step 1 Create a partitioned table and an ordinary table.

```
gaussdb=# CREATE TABLE t1(c1 INT,c2 INT) PARTITION BY RANGE (c2)
(
    PARTITION P1 values less than (5),
    PARTITION P2 values less than (10),
    PARTITION P3 values less than (101)
);
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100), generate_series(1,100));
gaussdb=# CREATE TABLE t1_exchange(c1 INT,c2 INT);
gaussdb=# INSERT INTO t1_exchange VALUES(generate_series(10,50), generate_series(10,50));
-- View the table.
gaussdb=# \d+ t1
                      Table "public.t1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off
gaussdb=# \d+ t1_exchange
                      Table "public.t1_exchange"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off
```

Step 2 Set the keyword **ONLINE** and perform related operations.

- Merge partitions online.

  ```
  gaussdb=# ALTER TABLE ONLINE t1 MERGE PARTITIONS p1, p2 into PARTITION p2_new;
  ALTER TABLE
  gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='t1'::regclass AND
  parttype <> 'r' order by 1;
   relname | boundaries |   oid
  ---------+------------+---------
   p2_new  | {10}       | 4397673
   p3      | {101}      | 4397634
  (2 rows)
  ```

- Splite a partition online.

  ```
  gaussdb=# ALTER TABLE ONLINE t1 SPLIT PARTITION p2_new AT (5) INTO (PARTITION P1,
  PARTITION P2);
  ALTER TABLE
  gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='t1'::regclass AND
  parttype <> 'r' order by 1;
   relname | boundaries |   oid
  ---------+------------+---------
   p1      | {5}        | 4397706
   p2      | {10}       | 4397707
   p3      | {101}      | 4397634
  (3 rows)
  ```

- Exchange partitions online.

  ```
  gaussdb=# ALTER TABLE ONLINE t1 EXCHANGE PARTITION (p3) WITH TABLE t1_exchange;
  ALTER TABLE
  gaussdb=# SELECT COUNT(*) FROM t1 PARTITION (p3);
   count
  -------
     41
  (1 row)
  ```

- Move partition tablespaces online.

  ```
  gaussdb=# CREATE TABLESPACE tbs RELATIVE LOCATION 'tablespace1/tbs';
  CREATE TABLESPACE
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# ALTER TABLE ONLINE t1 MOVE PARTITION p1 TABLESPACE tbs;
ALTER TABLE
gaussdb=# SELECT relname, spcname FROM pg_partition t1, pg_tablespace t2 WHERE
T1.reltablespace=t2.oid and t1.parentid='t1'::regclass;
 relname | spcname
---------+---------
 p1      | tbs
(1 row)
```

- In the preceding online DDL operations, concurrent read and write operations can be performed.

```
gaussdb=# INSERT INTO t1 VALUES(11, 11);
INSERT 0 1
gaussdb=# INSERT INTO t1_exchange VALUES(51, 51);
INSERT 0 1
```

Step 3 View a table definition.

```
gaussdb=# \d+ t1
                    Table "public.t1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off
gaussdb=# \d+ t1_exchange
                    Table "public.t1_exchange"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

gaussdb=# SELECT relname, reltablespace, boundaries FROM pg_partition;
 relname | reltablespace | boundaries
---------+---------------+------------
 t1      |             0 |
 p1      |       4397520 | {5}
 p2      |             0 | {10}
 p3      |             0 | {101}
(4 rows)
```

Step 4 Use VACUUM FULL to recycle partition space online.

```
gaussdb=# VACUUM FULL t1 PARTITION (p1) ONLINE;
VACUUM
```

----End

7.22 Bucket Distributed Tables

Overview

Bucket distributed tables are stored in segment-page mode and data is physically split by bucket based on the distribution mode specified by users.

Currently, bucket distributed tables can be Astore hash distributed tables and Ustore hash distributed tables. Replication tables, local/global temporary tables, and unlogged tables are not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Advantages of the Bucket Distributed Table

When data volume increases and requires cluster scale-out, you can precisely control the bucket files to be migrated and dynamically generate incremental log streams for these buckets. The incremental log streams are independent of system WALs and do not impact regular service flows, existing only during the scale-out phase.

7.22.1 Hash Bucket tables

Introduction to the Hash Bucket Table

In a hash bucket table, data is fragmented based on the hash value of the distribution key. Data with the same hash value is stored in the same group of bucket segment-page files. Each group of bucket files is completely decoupled.

7.22.1.1 Creating a Hash Bucket Table

Before creating the first hash bucket table in a database, you need to bind the current database to the default node group of the cluster.

Step 1 (Optional) Bind a node group to a database to generate the mapping between a hash distributed table on a CN and a bucket on a DN. The syntax is as follows:

```
ALTER DATABASE databasename TO GROUP groupname;
```

For details about the syntax, see "SQL Reference > SQL Syntax > > A > ALTER DATABASE" in *Reference* .

- Query the node group.

```
gaussdb=# select group_name from pgxc_group where is_installation = true;
 group_name
------------------------
 group_version1
(1 row)
```

- Bind the table to the default node group of the cluster. For details about the syntax, see "SQL Reference > SQL Syntax > A > ALTER DATABASE" and "SQL Reference > SQL Syntax > C > CREATE NODE GROUP" in *Reference*.

```
gaussdb=# ALTER DATABASE postgres TO GROUP group_version1;
ALTER DATABASE
```

  NOTE

  If no node group is bound before a hash bucket table is created for the first time, the table creation command is automatically bound to the default node group of the cluster.

```
gaussdb=# CREATE TABLE hashbucket_table1(c1 VARCHAR2 (30)) WITH(storage_type=ustore, hashbucket=on) distribute by hash(c1);
NOTICE:  This is the first time ever to create a hashbucket table in this database. The database will be automatically altered to the installation node group "group_version1".
NOTICE:  hashbucket table need segment storage, set segment to on by default
CREATE TABLE
```

Step 2 The syntax for creating a hash bucket table is the same as that for creating an ordinary table. When creating a table, you need to set **hashbucket** to **on** and explicitly or implicitly specify the hash distribution key.

- The default value for the storage parameter **hashbucket** is **off**. The value range of this parameter is as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **on**: The created table is a hash bucket table.
  - **off**: The created table is not a hash bucket table.
- Hash distribution key: At least one column must be specified as the hash distribution key for a hash bucket table, which determines the bucket for data distribution. The hash distribution key can be explicitly specified using the DISTRIBUTE BY hash(*column_name*) clause during table creation. If it is not explicitly specified, the first column that meets the hash distribution condition is implicitly chosen.

----End

- Create an ordinary hash bucket table.

```
gaussdb=# CREATE TABLE hashbucket_table1(c1 VARCHAR2 (30)) WITH(storage_type=ustore,
hashbucket=on) DISTRIBUTE BY hash(c1);
NOTICE:  hashbucket table need segment storage, set segment to on by default
CREATE TABLE
```

- Create a hash bucket partitioned table.

```
gaussdb=# CREATE TABLE hashbucket_partition_table_range1 (
    c1 int ,
    c2 bigint NOT NULL ,
    c3 VARCHAR2 ( 30 )
) WITH(storage_type=ustore, hashbucket=on) DISTRIBUTE BY hash(c3)
PARTITION BY RANGE(c1)(
PARTITION p1 VALUES LESS THAN (100000),
PARTITION p2 VALUES LESS THAN (200000),
PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
CREATE TABLE
```

For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE TABLE" and "SQL Reference > SQL Syntax > C > CREATE TABLE PARTITION" in *Reference*.

NOTE

- Creating the first hash bucket table in each database is slower than creating an ordinary table because it involves the creation of 1024 groups of segment-page files.
- When multiple tables are concurrently created, they will be successfully created only after the segment-page files are created for the first table.
- If the GUC parameter **enable_hashbucket** is set to **on**, you do not need to specify **WITH(hashbucket=on)**. By default, hash bucket tables are created. For details about the parameters, see "Database GUC Parameters > GUC Parameters > Miscellaneous Parameters" in *Reference*.

7.22.1.2 Bucket Indexes

The hash bucket table supports both the cross-bucket index (CBI) and local bucket index (LBI). For a hash bucket table, a CBI or LBI can be created by specifying the index storage parameter **crossbucket** as **on** or **off**. If **crossbucket** is not specified, an LBI is created by default.

7.22.1.2.1 Recommended Application Scenarios of Bucket Indexes

CBIs are stored in common segment-page mode, meaning that all bucket index tuples are created within a single unified index. LBIs are stored in hash-bucket segment-page mode, meaning that a subindex is created for each bucket.

LBIs implement intra-bucket indexing for hash bucket storage, while CBIs provide the cross-bucket indexing capability for hash bucket storage. When an index is

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

created, one index file can correspond to all bucket files at the table or partition level. CBIs enable cross-bucket index scan for tables of a specified granularity, preventing the need to sequentially scan a large number of bucket files in the current table before querying the target tuple and thereby enhancing query performance. Recommended application scenarios of CBIs and LBIs:

- When the hash distribution keys of service statements are all within the same bucket, or when the statement conditions include the hash distribution key and the scan results after pruning involve only a few buckets, intra-bucket indexing is required; in such cases, LBIs are recommended.
- When the conditions of a service statement do not include the hash distribution key or when a large number of buckets need to be scanned after filtering, cross-bucket indexing is required; in such cases, CBIs are recommended.

Support for Bucket Indexing

The kernel supports the following four combinations of the hash bucket partitioned table indexes.

Index Type	CBI	LBI
GPI	Yes	Yes
LPI	Yes	Yes

- Create an LBI for a hash bucket table.
```
gaussdb=# CREATE INDEX idx1 ON hashbucket_table1(c1) WITH(crossbucket=off);
CREATE INDEX
```
- Create a CBI for a hash bucket table.
```
gaussdb=# CREATE INDEX idx2 ON hashbucket_table2(c1) WITH(crossbucket=on);
CREATE INDEX
```
- Create an "LPI+LBI" combination for a hash bucket partitioned table.
```
gaussdb=# CREATE INDEX idx1_tb1 ON hashbucket_partiton_table_range1(c1) LOCAL
WITH(crossbucket=off);
CREATE INDEX
```
- Create an "LPI+CBI" combination for a hash bucket partitioned table.
```
gaussdb=# CREATE INDEX idx2_tb1 ON hashbucket_partiton_table_range1(c2) LOCAL
WITH(crossbucket=on);
CREATE INDEX
```
- Create a "GPI+LBI" combination for a hash bucket partitioned table.
```
gaussdb=# CREATE INDEX idx1_tb1 ON hashbucket_partiton_table_range1(c1) GLOBAL
WITH(crossbucket=off);
CREATE INDEX
```
- Create a "GPI+CBI" combination for a hash bucket partitioned table.
```
gaussdb=# CREATE INDEX idx2_tb1 ON hashbucket_partiton_table_range1(c2) GLOBAL
WITH(crossbucket=on);
CREATE INDEX
```

For details about the syntax, see "SQL Reference > SQL Syntax > C > INDEX" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.22.1.3 Feature Specifications and Limitations of the Hash Bucket Table

Limitations of the Hash Bucket Table

- Hash bucket tables support only distributed Astore/Ustore row store, hash distribution keys, and partitioned tables. They do not support RANGE, LIST, or MurmurHash distribution, or subpartitioned tables, temporary tables, unlogged tables, foreign tables, and replication tables.
- Hash bucket tables do not support GIN or GiST indexes.
- Hash bucket tables are bound to segment-page storage, that is, when **hashbucket** is set to **on**, **segment** must be set to **on**.
- Due to the addition of file storage information in the bucket dimension to the hash bucket table, it is necessary to appropriately increase the value of **segment_buffers** for performance considerations.
- Hash bucket tables do not support table-level backup for fine-grained backup and restoration, but support fine-grained table-level restoration, fine-grained database-level backup and restoration, cluster-level backup and restoration, and PITR.
- Hash bucket tables do not support online DDL operations and do not support M-compatible DDL and DML operations.
- You are advised not to create a small table (ordinary table: < 8 MB; partitioned table: < 8 MB x Number of partitions) as a hash bucket table; otherwise, the sequential scan performance deteriorates twice that of an ordinary table. Hash bucket tables utilize segment-page storage, and the physical file storage layer incorporates the bucket dimension. Consequently, empty tables or those with a minimal amount of data preoccupy a certain amount of storage space. Compared to ordinary tables, there is merely no space bloat for storage scenarios with over 10 GB of data on a single DN.
- No complete-refresh materialized view of the hash bucket type can be created. No fast-refresh materialized view can be created for a hash bucket table.
- Ustore hash bucket tables and Astore hash bucket tables cannot coexist in a database.
- For a hash bucket partitioned table with a GPI, when ALTER TABLE ... ADD/DROP/TRUNCATE/SPLIT PARTITION ... is executed, a few metadata pages of the deleted partition will remain. These residual pages can be deleted only when automatic clearing of residual segment-page data is enabled, that is, when the GUC parameter **enable_auto_segment_remain_cleanup** is set to **on** (the default value). You are advised to set this parameter to **on** when the hash bucket partitioned table involves the preceding DDL operations.
- Columns of the enumerated type, composite type, composite array type, table type, and special type ("regproc" "pg_node_tree" "regprocedure" "regoper" "regoperator" "regclass" "regtype" "_regprocedure" "_regoper" "_regoperator" "_regclass" "_regtype" "regconfig" "regdictionary" "_regconfig" "_regdictionary") cannot be used to create or alter hash bucket tables. Hash bucket tables that have been created and contain columns of the preceding types cannot be scaled out.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Limitations of the Ustore Hash Bucket Table

- The Ustore hash bucket table inherits all limitations of the hash bucket table.
- The Ustore hash bucket table inherits the limitations of the Ustore table. The ALTER TABLE MERGE PARTITIONS operation cannot be performed in transaction blocks or stored procedures. CBIs, LBIs, ordinary Ustore indexes, GLOBAL indexes, and LOCAL indexes can be created or rebuilt online in parallel. PCR indexes are not supported.
- The Ustore hash bucket table does not support batch insertion of UB-tree indexes.
- The Ustore hash bucket table does not support online creation of GSIs.
- The INCLUDE clause cannot be used during the creation of CBIs for a Ustore hash bucket table.
- The Ustore hash bucket table does not support table type conversion.

Feature Specifications of the Hash Bucket Table

The DML performance of a hash bucket table (ordinary table or partitioned table) deteriorates by less than 5% compared with that of an ordinary table (ordinary table or partitioned table).

7.22.1.4 Pruning a Hash Bucket Table

During the SELECT query on a hash bucket table, if the predicate of the SELECT statement is specified as the distribution key of the hash bucket table, the hash bucket table is pruned, greatly reducing the number of accessed buckets. The following is an example:

```
-- Prepare the environment.
gaussdb=# SET enable_fast_query_shipping=off;
gaussdb=# CREATE TABLE t1 (a int, b int, c int) WITH (hashbucket=on, storage_type=astore);
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 10), generate_series(1, 10));
gaussdb=# ANALYZE t1;
-- Display the execution plan.
gaussdb=# EXPLAIN (costs false,verbose on) SELECT count(*) FROM t1 WHERE a = 1;
                    QUERY PLAN
-----------------------------------------------------------
 Aggregate
   Output: pg_catalog.count(*)
   ->  Streaming (type: GATHER)
         Output: (count(*))
         Node/s: datanode1
         ->  Aggregate
               Output: count(*)
               ->  Seq Scan on public.t1
                     Output: a, b, c
                     Distribute Key: a
                     Filter: (t1.a = 1)
                     Selected Buckets 1 of 1024 : 325
(12 rows)
```

The plan shows that bucket 325 is selected. During actual execution, the data of the bucket is directly read, avoiding read on irrelevant buckets.

NOTE

The hash bucket plan can be displayed only after **EXPLAIN VERBOSE** is enabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameterized Path Pruning on a Hash Bucket Table

When a parameterized path plan is selected and the internal table is a hash bucket table, if the parameterized column is the distribution key of the hash bucket table, the parameterized path is pruned to dynamically reduce the number of buckets to be accessed.

```
-- Prepare the environment.
gaussdb=# SET enable_fast_query_shipping=off;
gaussdb=# CREATE TABLE t2 (a int, b int, c int) WITH (hashbucket=on, storage_type=astore);
gaussdb=# INSERT INTO t2 VALUES(generate_series(1, 10), generate_series(1, 10));
gaussdb=# CREATE INDEX index_t2 on t2(a, b);
gaussdb=# ANALYZE t2;
-- Display the execution plan.
gaussdb=# EXPLAIN (costs false,verbose on) SELECT /*+ nestloop_index(t2, index_t2)*/* FROM t1 JOIN t2
ON t1.b = t2.a;
                        QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.a, t1.b, t1.c, t2.a, t2.b, t2.c
   Node/s: All datanodes
   ->  Nested Loop
         Output: t1.a, t1.b, t1.c, t2.a, t2.b, t2.c
         ->  Streaming(type: REDISTRIBUTE)
               Output: t1.a, t1.b, t1.c
               Distribute Key: t1.b
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Seq Scan on public.t1
                     Output: t1.a, t1.b, t1.c
                     Distribute Key: t1.a
                     Selected Buckets 1024 of 1024 : all
         ->  Index Scan using index_t2 on public.t2
               Output: t2.a, t2.b, t2.c
               Distribute Key: t2.a
               Index Cond: (t2.a = t1.b)
               Selected Buckets 1024 of 1024 : all (ppi-pruning)
(19 rows)
```

You can search for "Selected Buckets" and check for "ppi-pruning" in the same line. If it exists, the plan will perform parameterized path pruning.

To know the pruning details, run the **EXPLAIN ANALYZE** statement on the DN.

```
-- Directly connect to a DN.
gaussdb=# EXPLAIN (costs false,verbose on,analyze on) SELECT /*+ nestloop_index(t2, index_t2)*/* FROM t1
JOIN t2 on t1.b = t2.a;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Nested Loop (actual time=4.743..10.928 rows=6 loops=1)
   Output: t1.a, t1.b, t1.c, t2.a, t2.b, t2.c
   ->  Seq Scan on public.t1 (actual time=4.607..10.471 rows=6 loops=1)
         Output: t1.a, t1.b, t1.c
         Selected Buckets 1024 of 1024 : all
   ->  Index Scan using index_t2 on public.t2 (actual time=0.124..0.132 rows=6 loops=6)
         Output: t2.a, t2.b, t2.c
         Index Cond: (t2.a = t1.b)
         Selected Buckets 1024 of 1024 : all (ppi-pruning) actual scan bucket num:6
 Total runtime: 23.424 ms
(10 rows)
```

In the preceding information, "actual scan bucket num:6" is found, indicating that six buckets are accessed.

Limitations

- The following bucket type is supported: hash bucket.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The following operator types are supported: index scan, index-only scan, and bitmap scan.
- The following expression types are supported: comparison expressions (<, <=, =, >=, >) and logical expression (AND).
- Subquery expressions, stable and volatile functions, cross-query block parameterized paths, and BitmapOr and BitmapAnd operators are not supported.
- Only local bucket indexes (LBIs) are supported. Cross bucket indexes are not supported.
- Only one distribution key is supported.
- When the hash bucket table is driven by an outer join, only the equal (=) condition is supported.

7.22.2 Best Practices for Bucket Distributed Tables

7.22.2.1 Setting segment_buffers

segment_buffers is used to set the memory size of segment-page metadata pages. To improve performance, you need to set a large buffer to ensure that metadata pages of active tables can be cached in the buffer and are not frequently replaced.

A hash bucket table, consisting of 1024 buckets, is stored based on the segment-page format. Each bucket is managed by a group of bucket segment-page files. On a DN, the metadata pages are mainly stored in bucket file 1 and the segment-page file 1. On a CN, they are mainly stored in segment-page file 1.

Metadata Pages of Bucket File 1

Three pages (main_fork, fsm_fork, and vm_fork) of bucket file 1 will be allocated for the following tables or indexes to store hash bucket metadata:

- Tables: ordinary tables, each partition of a partitioned table, TOAST tables, and TOAST indexes.
- LBIs: LBIs for ordinary tables, GLOBAL LBIs, and LOCAL LBI for each partition.

The number of pages can be estimated according to the following formula:

```
Number of buckets on a single DN x 3 x ((Number of ordinary tables + Number of partitions in a partitioned table) x 3 + Number of LBIs + Number of LOCAL LBI partitions)
```

In the formula, the first 3 indicates main_fork, fsm_fork, and vm_fork, and the second 3 indicates the table, TOAST table, and TOAST index. It is assumed that all user tables have TOAST tables and TOAST indexes for easier calculations.

Metadata Pages of Segment-Page File 1

One page of segment-page file 1 will be allocated for a table or index to store metadata. The number of pages can be estimated according to the following formula:

```
(Number of ordinary tables + Number of partitions in the partitioned tables) x 3 + Number of indexes + Number of LOCAL index partitions
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

In the formula, 3 indicates the table, TOAST table, and TOAST index. It is assumed that all user tables have TOAST tables and TOAST indexes for easier calculations.

An Example of Setting segment_buffers

Assume that there are 10 active tables, including four ordinary tables and six partitioned tables. The ordinary tables have 8 indexes, 5 of which are LBIs. Each partitioned table has 20 partitions and a total of 12 indexes, with 7 LOCAL LBIs. Assume that the cluster has three DN shards and 342 buckets are allocated to each DN.

The number of metadata pages can be estimated as follows:

- Bucket file 1:

```
Number of buckets on a single DN x 3 x ((Number of ordinary tables + Number of partitions in a partitioned table) x 3 + Number of ordinary table LBIs + Number of LOCAL LBI partitions)
=  342 * 3 * ((4 + 6 * 20) * 3 + 5 + 7 * 20)
= 530442 pages
≈ 4144MB
```

- Segment-page file 1:

```
(Number of ordinary tables + Number of partitions in the partitioned tables) x 3 + Number of indexes + Number of LOCAL index partitions
= (4 + 6 * 20) * 3 + (8 + 12) + 7 * 20
= 532 pages
≈ 4MB
```

In this example, when memory is sufficient, you are advised to set **segment_buffers** for each DN to a value of at least 4148 MB and for each CN to a value of at least 4 MB.

NOTE

- You are advised to set **segment_buffers** to a value greater than or equal to the total metadata page space of the active table to prevent performance deterioration due to metadata page buffer replacement.
- Hash bucket tables, especially partitioned tables, consume a significant number of metadata pages; you are advised to estimate and set **segment_buffers** based on the preceding formula.
- Note that the following condition must be met: **data_replicate_buffer_size** + **segment_buffers** + **shared_buffers** + **wal_buffers** + **temp_buffers** + **maintenance_work_mem** + **work_mem** + **query_mem** + **wal_receiver_buffer_size** (standby node) < **max_process_memory** < Memory size of the physical machine. If the total memory size of the physical machine is insufficient, you are advised to set **segment_buffers** to a value calculated using the following formula: Reference value of **shared_buffers** x Estimated metadata volume/Total data volume.
- Some DDL operations require loading a large number of metadata pages. For example, the "\d+" operation loads metadata pages for all tables. The loading process becomes time-consuming when metadata pages are loaded into the buffer for the first time or when the value of **segment_buffers** is insufficient.
- After setting the **segment_buffers** parameter, you need to restart the cluster for the setting to take effect.

7.22.2.2 Space Preallocation for a Bucket Distributed Table

The GUC parameter **enable_segment_datafile_preallocate** controls whether to enable immediate disk space allocation for segment-page file extension. When this parameter is set to **off**, space is reserved as holes for file extension only upon

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

actual use. When this parameter is set to **on**, space is preallocated for file extension. The default value is **on**.

NOTE

- If the disk space is limited, you are advised to disable **enable_segment_datafile_preallocate**.
- When **enable_segment_datafile_preallocate** is disabled, during full build or backup and restoration scenarios, after data is copied to the target node, the holes will be padded with 0 to occupy space, resulting in higher disk usage on the target node compared to the source node.

The space occupied by a bucket distributed table includes both metadata space and data space, involving five files (1 to 5) in total. File 1 is the bucket metadata file, and files 2 to 5 are data files.

For details about the calculation method of the metadata pages, see **Setting segment_buffers**. The data pages are related to the data volume and are extended as follows:

- Segment-page file 2:

  Each time a CBI is created, an extent (eight pages) is allocated to segment-page file 2, regardless of whether the current table contains data. If there is no available extent, the system will first extend the segment-page file with a step of 128 MB before space allocation.
- Bucket file 2:

  Each time an LBI or TOAST index is created for a hash bucket table, a new extent (eight pages) is allocated to file 2, regardless of whether the current table contains data. If there is no available extent, the system will first extend the segment-page file with a step of 8 MB before space allocation.

The preallocated space of bucket distributed tables is calculated as follows:

- **enable_segment_datafile_preallocate** is set to **off**.

Preallocated Space	Hash Bucket Table	Hash Bucket Partitioned Table
Number of heap pages	(Number of hash bucket tables + Number of TOAST tables) x 1024	(Total number of partitions in the hash bucket partitioned tables + Total number of partitions in the TOAST tables) x 1024
Number of index pages	(Number of CBIs in the hash bucket tables) x 2 + (Number of TOAST indexes + Number of LBIs in the hash bucket tables) x 1024 x 2 • **2**: one page from file 1 and one page from file 2. • **1024**: total number of buckets.	[(Number of TOAST indexes in the hash bucket partitioned table + Number of LOCAL LBIs) x 1024 + Number of LOCAL CBIs] x Number of partitions x 2 + Number of GLOBAL LBIs x 2 + Number of GLOBAL CBIs x 2 • **2**: one page from file 1 and one page from file 2. • **1024**: total number of buckets.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **enable_segment_datafile_preallocate** is set to **on**.

Preallocated Space	Hash Bucket Table	Hash Bucket Partitioned Table
Number of heap pages	(Number of hash bucket tables + Number of TOAST tables) x 1024	(Total number of partitions in the hash bucket partitioned tables + Total number of partitions in the TOAST tables) x 1024
Number of index pages	(Number of CBIs in the hash bucket tables) x 9 + (Number of TOAST indexes + Number of LBIs in the hash bucket tables) x 1024 x 9 • **9**: one page from file 1 and eight pages from file 2. • **1024**: total number of buckets.	[(Number of TOAST indexes in the hash bucket partitioned table + Number of LOCAL LBIs) x 1024 + Number of LOCAL CBIs] x Number of partitions x 9 + Number of GLOBAL LBIs x 9 + Number of GLOBAL CBIs x 9 • **9**: one page from file 1 and eight pages from file 2. • **1024**: total number of buckets.

Space Preallocation Example

- Ordinary table

  Assume that the table has one TOAST index, one LBI, and one CBI.

  – Space preallocation disabled: 16 MB (2 x 8 KB x 1024) for storing data;

    32 MB (2 x (8 KB + 8 KB) x 1024 + (8 KB + 8 KB)) for storing indexes.

    Total: 48 MB

  – Space preallocation enabled: 16 MB (2 x 8 KB x 1024) for storing data;

    144 MB (2 x (8 KB + 64 KB) x 1024 + (8 KB + 64 KB)) for storing indexes.

    Total: 232 MB

- Partitioned table (each partition is equivalent to one ordinary table)

  Assume that the table has 730 partitions, one primary key LOCAL index, one TOAST table, one TOAST index, one "LPI+LBI" combination, one "LPI+CBI" combination, one "GPI+CBI" combination, and one "GPI+LBI" combination.

  – Space preallocation disabled: 11 GB (2 x 730 x 8 KB x 1024) for storing data;

    134 GB (730 x [3 x (8 KB + 8 KB) x 1024 + (8 KB + 8 KB)] + (8 KB + 8 KB) + (8 KB + 8 KB) x 1024) for storing indexes.

    Total: 45 GB

  – Space preallocation enabled: 11 GB (2 x 730 x 8 KB x 1024) for storing data;

    154 GB (730 x [3 x (8 KB + 64 KB) x 1024 + (8 KB + 64 KB)] + (8 KB + 64 KB) + (8 KB + 64 KB) x 1024) for storing indexes.

    Total: 165 GB

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.23 Recompilation upon Invalidation

7.23.1 Overview

Description

Recompilation upon invalidation refers to the operation of recompiling a view object in a database after the object becomes invalid. It aims to restore the functionality of invalid objects and ensure that these objects can be correctly executed. Invalidation may be caused by various reasons, especially after the database architecture or dependent objects change, such as table definition modification, and deletion or update of dependent objects.

Recompilation upon invalidation involves the following functional modules:

- One-off import: creates a view. The view can be inserted into the system catalog of the database even though it does not meet the compilation requirements during pre-check.
- Cascading invalidation: When an object is changed, the objects depending on it become invalid. For example, when a table column is updated, the object **obj1** that depends on the current table becomes invalid, and the object **obj2** that depends on the object **obj1** also becomes invalid.
- Recompilation: includes manual recompilation, automatic recompilation, and batch recompilation.
  - Manual recompilation:
    ```
    ALTER VIEW objname COMPILE;
    ```
  - Automatic recompilation:

    GaussDB automatically attempts to recompile invalid objects. For example, when an invalid view is called, the database automatically recompiles the view during execution.
  - Batch recompilation:

    You can run the following statement to query all invalid objects in the database:

    ```
    SELECT * FROM pg_object WHERE valid='f';
    ```

    You can run the following statement to call the API to recompile all invalid objects in batches:

    ```
    CALL PG_CATALOG.GS_COMPILE_SCHEMA(schema_name, false, 10);
    ```

To perform recompilation upon invalidation, you need to enable the following GUC parameters:

- Set **enable_force_create_obj** to **on** to enable the one-off import functionality for views.
- Set **enable_view_invalidation** to **on** to enable the cascading invalidation functionality for views. If this parameter is set to **off**, the functionality is disabled for views.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenarios

Scenario 1: The database is deployed for the first time.

Scenario 2: The user service changes.

Specifications and Limitations

Recompilation upon invalidation has the following limitations:

- Compatibility: Databases in any compatible mode are supported.
- Specifications:
  - When a view is created and a partition is specified, the dependency relationship between the view and the partition is added. When a view is created and a user-defined or CAST type conversion is performed, the dependency between the view and the user-defined or CAST type conversion is added.

    NOTE

    For details about dependency relationship types, see "System Catalogs and System Views > System Catalogs > Other System Catalogs > PG_DEPEND" in *Reference*.
  - If an object on which a view depends is restored after the transaction is started, the view cannot be automatically recompiled by calling the view.
- Functionality restrictions:
  - The standby node does not support recompilation upon invalidation.
  - DDL changes to synonyms cannot trigger view invalidation.
  - Views that access remote synonyms through database links are not affected by remote synonym changes.
  - Temporary views and views that depend on local temporary tables cannot be automatically recompiled. Only manual recompilation is supported.
  - Views in the recycle bin cannot be recompiled upon invalidation.

Technical Architecture

Recompilation upon invalidation relies on system catalogs and syntaxes.

- System catalogs:
  - PG_OBJECT records the object validity information.
  - PG_DEPEND records the dependency relationships between database objects.
  - GS_DEP_SOURCE records the original definitions and some compilation products of objects.
- Syntaxes involve ALTER VIEW objname COMPILE, view creation, and view access syntaxes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.23.2 Technical Principle

Core Mechanism

- Dependency record

  Dependency relationships between database objects are recorded to implement cascading invalidation of database objects.

- Cascading invalidation

  When an object is changed, the objects depending on it along the dependency chain become invalid.

- Recompilation

  Database objects can be updated to be valid through automatic recompilation or manual recompilation.

API Description

Table 7-28 Invalidation and Recompilation APIs

APIs for recompilation upon invalidation	
Syntaxes	ALTER VIEW view_name COMPILE;
System catalogs	• PG_DEPEND records the dependency relationships between database objects. • GS_DEP_SOURCE records the original definitions and some compilation products of objects. • PG_OBJECT records the object status information.
GUC parameters	• If **enable_force_create_obj** is set to **on**, the one-off import functionality for views is enabled. • If **enable_view_invalidation** is set to **on**, the cascading invalidation functionality for views is enabled.
Functions	PG_CATALOG.GS_COMPILE_SCHEMA recompiles views in a specified schema. If an error is reported during view compilation, the error will be captured and other objects will be compiled until all objects are successfully compiled or the number of compilation attempts reaches the maximum.

7.23.3 Operation Guide

Prerequisites

- To disable the cascading invalidation of views, set the GUC parameter **enable_view_invalidation** to **on**.
- To import views to a database at a time, set the GUC parameter **enable_force_create_obj** to **on**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

You can use the following syntax to recompile a view:

```
ALTER VIEW view_name COMPILE;
```

Precautions

- Before using the functionality, check whether invalid objects exist. If invalid objects are called, exceptions may occur.
- Avoid constructing a scenario where objects depend on each other. Otherwise, exceptions may occur during access and compilation.
- During recompilation upon invalidation, do not perform other DML operations.
- If view recompilation fails, you can use the CREATE OR REPLACE VIEW syntax to rebuild the view.

Procedure

The following outlines the overall procedure. For specific code, refer to **Examples**.

Step 1 Manually create view objects in the database.

Step 2 Perform DDL changes on the objects on which the view depends.

Step 3 Check whether invalid objects exist in the PG_OBJECT system catalog. The value of the **valid** column of invalid objects is **false**.

Step 4 Run the **ALTER COMPILE** command or call pg_catalog.gs_compile_schema to recompile the objects.

Step 5 Check whether the **valid** column in the PG_OBJECT system catalog is **false**. Ensure that all objects become valid.

Step 6 Run services.

----End

Examples

- Example 1: Manually recompiling a view

```
-- Enable recompilation upon invalidation.
gaussdb=# SET enable_view_invalidation=on;

-- Create a table on which the view depends.
gaussdb=# CREATE TABLE tetab_v1(a int,b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE VIEW teview1 AS SELECT a, b FROM tetab_v1;
-- Drop the table.
gaussdb=# DROP TABLE tetab_v1;
-- Check the view's validity.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'teview1';
 relname | valid
---------+-------
 teview1 | f
(1 row)
-- Rebuild the table on which the view depends.
gaussdb=# CREATE TABLE tetab_v1(a int,b int);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Manually compile the view and check the object's validity again.
gaussdb=# ALTER VIEW teview1 COMPILE;
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'teview1';
 relname | valid
---------+-------
 teview1 | t
(1 row)
-- Drop the view.
gaussdb=# DROP VIEW teview1;
-- Drop the table.
gaussdb=# DROP TABLE tetab_v1;
```

- Example 2: Automatically recompiling a view

```
-- Enable recompilation upon invalidation.
gaussdb=# SET enable_view_invalidation=on;

-- Create a table on which the view depends.
gaussdb=# CREATE TABLE tetab_v2(a int,b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE VIEW teview2 AS SELECT a, b FROM tetab_v2;
-- Drop the table.
gaussdb=# DROP TABLE tetab_v2;
-- Check the view's validity.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'teview2';
 relname | valid
---------+-------
 teview2 | f
(1 row)
-- Rebuild the table on which the view depends.
gaussdb=# CREATE TABLE tetab_v2(a int,b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Perform automatic view recompilation and check the object's validity again.
gaussdb=# SELECT * FROM teview2;
 a | b
---+---
(0 rows)
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'teview2';
 relname | valid
---------+-------
 teview2 | t
(1 row)
-- Drop the view.
gaussdb=# DROP VIEW teview2;
-- Drop the table.
gaussdb=# DROP TABLE tetab_v2;
```

- Example 3: Recompiling views in batches

```
-- Enable recompilation upon invalidation.
gaussdb=# SET enable_view_invalidation=on;

-- Create a schema, a table, a view that depends on the table, and a view that depends on the view.
gaussdb=# DROP SCHEMA IF EXISTS tesch2 CASCADE;
gaussdb=# CREATE SCHEMA tesch2;
gaussdb=# SET current_schema = tesch2;
gaussdb=# CREATE TABLE tab1 (c1 INT, c2 INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE VIEW v_tab1 AS SELECT * FROM tab1;
gaussdb=# CREATE VIEW v_tab2 AS SELECT * FROM v_tab1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE VIEW v_tab3 AS SELECT * FROM tab1;
-- Drop the table.
gaussdb=# DROP TABLE tab1;
-- Check the object's validity.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'v_tab1' or c.relname = 'v_tab2' or c.relname = 'v_tab3';
 relname | valid
---------+-------
 v_tab1  | f
 v_tab2  | f
 v_tab3  | f
(3 rows)
-- Restore the dropped table.
gaussdb=# CREATE TABLE tab1 (c1 INT, c2 INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Recompile view objects in the tesch2 namespace in batches.
gaussdb=# CALL pg_catalog.gs_compile_schema('tesch2', true, 1);
INFO:  successful
CONTEXT:  referenced column: gs_compile_schema
SQL statement "select pkg_util.gs_compile_schema(:schema_name, :compile_all, :retry_times);"
PL/pgSQL function gs_compile_schema(character varying,boolean,integer) line 24 at EXECUTE
statement
INFO:  The number of executions is 1
INFO:  successful
 gs_compile_schema
-------------------
 t
(1 row)
-- Check the view's validity again.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'v_tab1' or c.relname = 'v_tab2' or c.relname = 'v_tab3';
 relname | valid
---------+-------
 v_tab1  | t
 v_tab2  | t
 v_tab3  | t
(3 rows)
-- Drop views.
gaussdb=# DROP VIEW v_tab3;
gaussdb=# DROP VIEW v_tab2;
gaussdb=# DROP VIEW v_tab1;
-- Drop the table.
gaussdb=# DROP TABLE tab1;
gaussdb=# RESET current_schema;
-- Drop the schema.
gaussdb=# DROP SCHEMA tesch2 CASCADE;
```

7.23.4 FAQ

Q: How do I check whether the recompilation upon invalidation feature is available?

A: Run the following command. In the command output, the value **on** of **enable_view_invalidation** indicates cascading invalidation and the value **on** of **enable_force_create_obj** indicates one-off data import.

```
gaussdb=# SELECT name,setting FROM pg_settings WHERE name in ('ddl_invalid_mode',
'enable_view_invalidation','enable_force_create_obj');
           name            | setting
---------------------------+---------
 enable_force_create_obj   | on
 enable_view_invalidation | on
(2 rows)
```

Q: How do I query the object validity?

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

A: To check the validity of a specified object, you can query the OID of the object from the related system catalog and query the **valid** column of pg_object based on the OID. If the value of **valid** is **true**, the object is valid. If the value is **false**, the object is invalid. The following is the example view information:

```
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'b_teview1';
  oid
--------
 221209
(1 row)
-- The view b_teview1 is valid.
gaussdb=# SELECT object_type, valid FROM pg_object WHERE object_oid = 221209;
 object_type | valid
-------------+-------
 v           | t
(1 row)
```

Q: How do I recompile objects?

A: You can recompile objects in the following ways:

- Automatic recompilation: For example, if view **view1** depends on table **tab1** and **tab1** is changed, **view1** becomes invalid. When **view1** is called after **tab1** is restored, it is automatically recompiled.
- Manual recompilation: For example, if view **view1** depends on table **tab1** and **tab1** is changed, **view1** becomes invalid. After **tab1** is restored, you can run **ALTER VIEW view1 COMPILE** to recompile the view.
- Batch recompilation: This method applies to scenarios where there are a large number of objects with complex dependencies. You can run the following statement to compile objects in a specified schema in batches.
  ```
  CALL pg_catalog.gs_compile_schema(schema_name, false, 10);
  ```

Examples

- Example 1: After a transaction is started and the object on which the view depends is restored, the view cannot be automatically recompiled by calling.
  ```
  -- Configure the following parameters:
  gaussdb=# SET enable_view_invalidation=on;

  -- Create a table.
  gaussdb=# CREATE TABLE base_tab1(a int, b int);
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  -- Create a view that depends on the table.
  gaussdb=# CREATE VIEW base_view1 AS SELECT a, b FROM base_tab1;
  -- Drop the table. The view becomes invalid.
  gaussdb=# DROP TABLE base_tab1;
  -- Check the view's validity.
  gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
  c.oid = o.object_oid WHERE c.relname = 'base_view1';
    relname   | valid
  ------------+-------
   base_view1 | f
  (1 row)
  -- Enable the view and restore the table on which the view depends in the transaction.
  gaussdb=# BEGIN;
  gaussdb=# CREATE TABLE base_tab1(a int, b int);
  NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
  HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
  CREATE TABLE
  -- Use the view to trigger automatic recompilation. The recompilation fails.
  gaussdb=# SELECT * FROM base_view1;
  ERROR:  View "base_view1" is invalid and please check view definition.
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
LINE 1: select * from base_view1;
                      ^
-- Roll back the transaction.
gaussdb=# ROLLBACK;
-- Enable the view again, restore the table on which the view depends in the view, and manually
recompile the view.
gaussdb=# BEGIN;
gaussdb=# CREATE TABLE base_tab1(a int, b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# ALTER VIEW base_view1 COMPILE;
-- Commit the view.
gaussdb=# COMMIT;
-- Check the view's validity again.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'base_view1';
  relname   | valid
------------+-------
 base_view1 | t
(1 row)
-- Drop the view.
gaussdb=# DROP VIEW base_view1;
-- Drop the table.
gaussdb=# DROP TABLE base_tab1;
```

- Example 2: DDL changes to synonyms cannot trigger view invalidation.

```
-- Configure the following parameters:
gaussdb=# SET enable_view_invalidation=on;

-- Create a table.
gaussdb=# CREATE TABLE base_tab2(a int, b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Create a synonym for the table.
gaussdb=# CREATE SYNONYM syn_tab FOR base_tab2;
-- Create a view that depends on the synonym.
gaussdb=# CREATE VIEW base_view2 AS SELECT a, b FROM syn_tab;
-- Drop the synonym. The synonym cannot be dropped because the view depends on it.
gaussdb=# DROP SYNONYM syn_tab;
ERROR:  synonym syn_tab cannot be dropped because other objects depend on it.
DETAIL:  view base_view2 depends on synonym public.syn_tab
HINT:  Use DROP ... CASCADE to drop the dependent objects.
-- Drop the view.
gaussdb=# DROP VIEW base_view2;
-- Drop the synonym.
gaussdb=# DROP SYNONYM syn_tab;
-- Drop the table.
gaussdb=# DROP TABLE base_tab2;
```

- Example 3: Temporary views cannot be automatically recompiled. They can only be recompiled manually.

```
-- Configure the following parameters:
gaussdb=# SET enable_view_invalidation=on;

-- Create a table.
gaussdb=# CREATE TABLE base_tab3(a int, b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Create a temporary view.
gaussdb=# CREATE TEMP VIEW temp_view AS SELECT a, b FROM base_tab3;
-- Drop the table.
gaussdb=# DROP TABLE base_tab3;
-- Check the view's validity.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'temp_view';
  relname  | valid
-----------+-------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 temp_view | f
(1 row)
-- Rebuild the table on which the view depends.
gaussdb=# CREATE TABLE base_tab3(a int, b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
-- Use the view to trigger automatic recompilation. An error is reported during recompilation.
gaussdb=# SELECT * FROM temp_view;
ERROR:  View "temp_view" is invalid and please check view definition.
LINE 1: SELECT * FROM temp_view;
                      ^
-- Manually recompile the view.
gaussdb=# ALTER VIEW temp_view COMPILE;
-- Check the view's validity again.
gaussdb=# SELECT c.relname, o.valid FROM pg_catalog.pg_object o JOIN pg_catalog.pg_class c ON
c.oid = o.object_oid WHERE c.relname = 'temp_view';
  relname  | valid
-----------+-------
 temp_view | t
(1 row)
-- Drop the view.
gaussdb=# DROP VIEW temp_view;
-- Drop the table.
gaussdb=# DROP TABLE base_tab3;
```

7.24 Locating Stored Procedure Performance Problems

Choose either of the following methods to locate stored procedure performance problems: For a stored procedure that is being executed, observe the statements that are being executed by the stored procedure to locate the blocking problem; for an executed stored procedure, query related system catalogs and system views to obtain the execution records and confirm the slow scenario.

In non-multi-primary mode, both scenarios are supported.

7.24.1 Locating Performance Problems During Stored Procedure Execution

If a stored procedure is blocked for a long time, you can use the methods provided in this section to analyze the performance.

7.24.1.1 Querying pg_stat_activity to View the Statements That Are Being Executed in a Stored Procedure

You can query pg_stat_activity to view the SQL statements that are being executed by a session. If a stored procedure is being executed by the session, you can check the statements that are being executed in the stored procedure.

7.24.1.1.1 Overview

Description

The **query** column in the pg_stat_activity system view displays the SQL statements that are being executed by the current session. This function is supported in versions later than V500R002C10.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenarios

If a session keeps executing a stored procedure for a long time, you can view the pg_stat_activity system catalog to determine the phase and statements of the stored procedure that is being executed by the current session.

Specifications and Limitations

- The GUC parameters **track_activities** and **track_procedure_sql** must be set to **on**.
- Considering database performance, the following types of statements can be viewed: variable assignment, IF, ELSEIF, CASE, OPEN CURSOR, FETCH CURSOR, FORCURSOR, dynamic statements, LOOP, WHILE, FOR loop, PERFORM, DML statements, nested subprocedures, COMMIT/ROLLBACK, and SAVE POINT.
- If a statement that cannot be recorded is executed, the **query** column in the pg_stat_activity system view displays the last statement that can be recorded.

Technical Architecture

Records in pg_stat_activity are collected by session. That is, the statements that are being executed by each session are collected. In the stored procedure scenario, the **query** column records the top-level statement called by a stored procedure by default. When a statement in the stored procedure is executed, if the statement can be recorded, the query information that is being executed by the current session is overwritten according to the statement processing logic.

7.24.1.1.2 Technical Principle

Core Mechanism

The pg_stat_activity system view can display the statements that are being executed. During statement parsing, APIs are called to write the statement information to the **query** column. To record the statements that are being executed in a stored procedure, the value of **query** in pg_stat_activity is overwritten by content in the following format: *query string*+*current level*+*lineno*+*execute stmt*, where **query string** indicates the top-level statement executed, **current level** indicates the current statement directly called, **lineno** indicates the line number of the current statement in the stored procedure, and **execute stmt** indicates the statement that is being executed.

Performance Metrics

If **track_procedure_sql** is enabled to record the SQL statements that are being executed in a stored procedure, the performance deteriorates by about 5%.

API Description

APIs for recording statements that are being executed in a stored procedure in pg_stat_activity	
System catalog	pg_stat_activity

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

APIs for recording statements that are being executed in a stored procedure in pg_stat_activity	
GUC parameters	• **track_activities** specifies whether pg_stat_activity records information. • **track_procedure_sql** specifies whether to record the statements that are being executed in a stored procedure. By default, it is enabled.

7.24.1.1.3 Operation Guide

Prerequisites

The **track_activities** and **track_procedure_sql** parameters have been set to **on**.

Syntax

N/A

Precautions

The length of the **query** column in the pg_stat_activity system view is controlled by the GUC parameter **track_activity_query_size** and is 1024 bytes by default. If any statement is too long, it will be truncated, causing incomplete display.

Examples

```
-- Create a stored procedure by referring to the following example.
gaussdb=# CREATE OR REPLACE PROCEDURE p1()
AS
BEGIN
pg_sleep(2);
END;
/
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE p2()
AS
BEGIN
pg_sleep(2);
END;
/
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE p3()
AS
BEGIN
p1();
p2();
END;
/
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE p4()
AS
BEGIN
p1();
p2();
END;
/
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE p5()
AS
BEGIN
p3();
p4();
END;
/
CREATE PROCEDURE
-- Session A calls the stored procedure.
gaussdb=# CALL p5();
-- Session B views the SQL statements that are being executed by session A during the execution of session A. After session A is executed, the last statement executed by session A will be recorded.
gaussdb=# SELECT query FROM pg_stat_activity WHERE query LIKE '%call%' AND sessionid != pg_current_sessid();
                                    query
-------------------------------------------------------------------------------------
 query string: call p5();, current level: p2(), lineno: 3, execute stmt: CALL pg_sleep(2)
(1 row)
```

7.24.1.2 Tracing Statements Executed by a Stored Procedure

7.24.1.2.1 Overview

Description

In versions later than 505.2.0, SQL statements executed in a stored procedure can be traced. After the functionality is enabled, the database records each statement executed during the execution of the stored procedure in real time.

Scenarios

During the execution of a stored procedure, you can view the execution status of the stored procedure through pg_stat_activity. By enabling the trace function, you can monitor the execution of the stored procedure in real time and further analyze the cause of slow execution based on the execution plan of statements in the stored procedure.

Specifications and Limitations

- Optional values of **level** in pl_trace_func_start are as follows: **session**, **portal**, **time**, and **threshold**, which are case-insensitive.
- The value range of **max_num** in pl_trace_func_start is 100 to 10000.
- The value of **threshold** in pl_trace_func_start must be greater than or equal to 0, in milliseconds. The **threshold** parameter takes effect only when **level** is set to **threshold**.
- If the session specified by the **session_id** parameter of pl_trace_func_start does not exist, an error will be reported.
- If the session specified by the **session_id** parameter of pl_trace_func_start has been traced and is the same as the session of the function to be called, a warning will be reported and the **trace** parameter is changed to the input parameter of this query.
- If the session specified by the **session_id** parameter of pl_trace_func_start has been traced and is different from the session of the function to be called, an error will be reported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If the **session_id** parameter in pl_trace_func_end is not specified, an error is reported. If the session is not traced, an error is reported.
- If the value of **session_id** in pl_trace_func_clean is **null**, all trace information is cleared. If the trace information contains data of the session specified by **session_id**, the data is cleared. If the trace information does not contain data of the session specified by **session_id**, an error will be reported.
- The pl_trace_func_clean function cannot delete data of a session that is being traced. If the session is being traced, the pl_trace_func_clean function skips it and reports a warning.
- Statements that have been executed are recorded even if an error is reported when the trace functionality is executed.
- The trace functionality can record trace information about a maximum of 100 sessions at a time.
- The trace functionality can be used only in thread pool mode (**enable_thread_pool** set to **on**).
- The trace functionality can be used only after a stored procedure has been executed for the traced session.
- The trace functionality does not support anonymous block statements or instantiated package statements.
- The trace functionality does not support the display of internal statements of nested subprogram, but displays the subprogram line number and subprogram function name.
- The trace functionality does not support stored procedures in dbe_perf mode.

Technical Architecture

In GaussDB, the PL/SQL statement execution process can be divided into the following processes: function execution, statement block execution, and statement execution. Callback functions are added before and after each specific statement for execution, so as to monitor the statement that is being executed, record the statement, and collect statistics.

7.24.1.2.2 Technical Principle

Core Mechanism

Tracing statements that are being executed in a stored procedure relies on the callback functions set during the stored procedure execution. The callback functions record and collect statements and statistics. Before a stored procedure is executed, you can call the dbe_perf.pl_trace_func_start API for the database to set the trace parameters and callback functions and enable the trace functionality. During the execution of the stored procedure, the callback functions before and after each statement store the records and information of the corresponding statement in a linked list. After the stored procedure is executed, you can call the dbe_perf.pl_trace_func_end API to clear the trace parameters and disable the trace functionality and call the dbe_perf.pl_get_trace_info API to read data from the linked list and display information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Performance Metrics

The **max_num** parameter of the dbe_perf.pl_trace_func_start function controls the maximum number of trace records, ranging from 100 to 10000. If the value is **10000**, the performance deteriorates by about 5%.

API Description

APIs for tracing statements executed in a stored procedure	
Functions	• DBE_PERF.PL_TRACE_FUNC_START(session_id, max_num, level, threshold): enables the trace functionality. • DBE_PERF.PL_TRACE_FUNC_END(session_id): disables the trace functionality. • DBE_PERF.PL_TRACE_FUNC_CLEAN(session_id): cleans the recorded trace information. • DBE_PERF.PL_GET_TRACE_INFO(session_id, level, database, func, source, plan, time, threshold): displays the recorded trace information.

7.24.1.2.3 Operation Guide

Prerequisites

The functionality can be used only in thread pool mode (**enable_thread_pool** set to **on**).

Syntax

N/A

Precautions

- PL_TRACE_FUNC_CLEAN cannot clean data on sessions that are executing the trace function.
- The trace functionality does not support stored procedures in DBE_PERF mode.

Examples

```
-- Set enable_thread_pool to on to enable the thread pool mode.
-- Create a stored procedure and execute it.
gaussdb=# DROP TABLE IF EXISTS t1;
NOTICE:  table "t1" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE t1(a INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE p1()
AS
BEGIN
FOR i IN 1..10 LOOP
INSERT INTO t1 VALUES (1);
END LOOP;
END;
/
CREATE PROCEDURE
-- Enable the trace functionality in the current session.
gaussdb=# CALL DBE_PERF.PL_TRACE_FUNC_START(pg_current_sessid());
 pl_trace_func_start
---------------------

(1 row)
-- Execute the stored procedure.
gaussdb=# CALL p1();
 p1
----

(1 row)
-- View the obtained trace information.
gaussdb=# SELECT * FROM dbe_perf.pl_get_trace_info();
 session_id |  level  | database | func  |               source                | plan | time | threshold
------------+---------+----------+-------+-------------------------------------+------+------+-----------
          2 | SESSION | postgres | 17542 | line 3, for i in 1..10 loop         |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
          2 | SESSION | postgres | 17542 | line 4, insert into t1 values (1); |      |      |
(11 rows)
-- Disable the trace function.
gaussdb=# CALL dbe_perf.pl_trace_func_end(pg_current_sessid());
 pl_trace_func_end
-------------------

(1 row)
-- Delete the recorded information.
gaussdb=# CALL dbe_perf.pl_trace_info_clean(pg_current_sessid());
 pl_trace_info_clean
---------------------

(1 row)
```

7.24.2 Locating Performance Problems After Stored Procedure Execution

In non-multi-primary mode, after a stored procedure is executed, you can locate and analyze the problems related to the stored procedure by referring to this section.

7.24.2.1 Slow SQL Statements in a Stored Procedure

You can view slow SQL statements in a stored procedure by querying related system catalogs and views, such as DBE_PERF.STATEMENT_HISTORY and DBE_PERF.STATEMENT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.24.2.1.1 Overview

Description

In 503.1.SPC1800 and later versions, slow SQL statements in a stored procedure can be analyzed by using DBE_PERF.STATEMENT_HISTORY and DBE_PERF.STATEMENT.

Scenarios

If a slow statement is found and the stored procedure calling the statement cannot be determined, you can query information such as the outermost call, performance of a single execution, and cost of executing the specific statement.

Specifications and Limitations

- The **instr_unique_sql_track_type** parameter must be set to **'all'**, so that DBE_PERF.STATEMENT_HISTORY and DBE_PERF.STATEMENT can record statements executed in stored procedures.
- To query DBE_PERF.STATEMENT_HISTORY, you must have the SYSADMIN or MONITOR ADMIN permission. The result can be queried only in the system database but cannot be queried in the user database.
- The DBE_PERF.STATEMENT_HISTORY system catalog is controlled by **track_stmt_stat_level**. The default value is **"OFF,L0"**, where the first part controls full SQL statements and the second part controls slow SQL statements. Records can be recorded in different columns of the table at different levels. The information recorded in some columns varies according to the level.
- To query the DBE_PERF.STATEMENT view, you must have the SYSADMIN or MONADMIN permission. You can view all statistics about normalized SQL statements received by a CN and related DNs on the CN, whereas you can view only the statistics about normalized SQL statements executed on the current node on a DN.
- To access WDR snapshot data, you must have the SYSADMIN or MONADMIN permission. WDRs can be generated only when the GUC parameter **enable_wdr_snapshot** is set to **on** and the number of snapshots is greater than or equal to 2.
- The statements in a stored procedure that can be recorded include DML statements, dynamic statements, PERFORM, OPEN CURSOR, and FETCH CURSOR. The value of **query** of the OPEN and FETCH statements will be converted to the query statement actually executed by the cursor.

Technical Architecture

In GaussDB, slow SQL statements refer to statements whose execution time exceeds the threshold specified by **log_min_duration_statement**.

Slow SQL statements of stored procedures are recorded by DBE_PERF.STATEMENT_HISTORY and DBE_PERF.STATEMENT.

- For the DBE_PERF.STATEMENT_HISTORY system catalog, a record is generated each time a statement is executed. Whether the record is updated to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DBE_PERF.STATEMENT_HISTORY depends on the value of **track_stmt_stat_level** and whether the current statement is a slow SQL statement.

- DBE_PERF.STATEMENT records normalized information, relying on a hash table to implement. Statements with the same **unique_query_id**, **parent_unique_sql_id**, **userid**, and **dbid** are normalized to the same record.

7.24.2.1.2 Technical Principle

Core Mechanism

Currently, **unique_query_id** can be generated before a statement is executed, and a record is inserted into DBE_PERF.STATEMENT for normalization, reporting, and update. During statement execution, the performance indicators generated for the statement are updated in DBE_PERF.STATEMENT. After the statement is executed, the system determines whether the statement is a slow SQL statement based on the value of **track_stat_stmt_level** and determines whether to record the statement in DBE_PERF.STATEMENT_HISTORY.

In the stored procedure scenario, the **unique_sql_id** of each statement called in a stored procedure is saved before the internal statement is executed. The **unique_sql_id** is used as the **parent_unique_sql_id** of the statement in the stored procedure. The calling relationship between statements is maintained based on the values of **parent_unique_sql_id** and **unique_sql_id**. When a statement in a stored procedure is executed, a record is inserted into DBE_PERF.STATEMENT. After each statement is executed, the system determines whether to record the statement in DBE_PERF.STATEMENT_HISTORY based on whether the statement is executed slowly and the value of **track_stmt_stat_level**. In contrast to the case of statements not called by a stored procedure, the performance indicators recorded in the DBE_PERF.STATEMENT table are updated only after the entire stored procedure is executed.

Performance Metrics

If **instr_unique_sql_track_type** is set to **all**, the execution duration of the stored procedure deteriorates by about 5% compared with that when **instr_unique_sql_track_type** is set to **top**.

API Description

APIs for diagnosing slow SQL statements in a stored procedure	
Function	DBE_PERF.GET_FULL_SQL_BY_PARENT_ID_AND_TIMESTAMP recursively queries the execution records of the statement and all substatements in DBE_PERF.STATEMENT_HISTORY within the execution time period based on the specified **parent_unique_sql_id**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

APIs for diagnosing slow SQL statements in a stored procedure	
System catalog	**parent_unique_sql_id** in DBE_PERF.STATEMENT_HISTORY indicates the **unique_query_id** of the stored procedure that calls the statement. The value is **0** for the top-level statement.
System view	**parent_unique_sql_id** in DBE_PERF.STATEMENT indicates the **unique_query_id** of the stored procedure that calls the statement. The value is **0** for the top-level statement.
GUC parameter	If **instr_unique_sql_track_type** is set to **all**, slow SQL statements executed in a stored procedure will be recorded.

7.24.2.1.3 Operation Guide

Prerequisites

Parameter **instr_unique_sql_track_type** has been set to **all**.

Syntax

N/A

Precautions

The maximum number of records in the DBE_PERF.STATEMENT table is controlled by **instr_unique_sql_count**. The default value of **instr_unique_sql_track_type** is **all**, recording statements in stored procedures by default. If this value is used, the number of statements recorded in the table is more likely to reach the threshold. After the threshold is reached, no new record will be generated. When the data volume reaches the threshold, you can use the reset_unique_sql(scope, clean_type, clean_value) API to clear the data in the table.

Query the DBE_PERF.STATEMENT_HISTORY table only in the system database. No data is stored in other databases.

Examples

```
-- Check the value of instr_unique_sql_track_type and ensure that the value is all.
gaussdb=# SHOW instr_unique_sql_track_type;
gaussdb=# SET instr_unique_sql_track_type='all';
-- For easy observation, temporarily set an L0 indicator to fully record SQL statements.
gaussdb=# SET track_stmt_stat_level='L0,L0';
-- Create a test database.
gaussdb=# CREATE DATABASE tedb;
-- Create a stored procedure and execute it.
gaussdb=# CREATE TABLE test(a INT,b INT);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
gaussdb=# INSERT INTO test VALUES(generate_series(1,30),2);
gaussdb=# CREATE OR REPLACE PROCEDURE test_proc1()
AS
num INT;
BEGIN
SELECT COUNT(*) INTO num FROM test;
END;
/
gaussdb=# CALL test_proc1();
 test_proc1
------------


(1 row)
-- Recursively query records in DBE_PERF.statement. The time-related indicators in the query result are affected by the actual environment and are not fixed.
gaussdb=# WITH p AS (SELECT * from DBE_PERF.STATEMENT WHERE query LIKE 'call test_proc1%' LIMIT 1)
SELECT
query,
n_soft_parse,
n_hard_parse,
n_returned_rows,
db_time,
cpu_time,
execution_time,
parse_time,
plan_time,
rewrite_time,
pl_execution_time,
pl_compilation_time,
data_io_time
FROM (
SELECT * FROM p UNION ALL SELECT * from DBE_PERF.STATEMENT
WHERE parent_unique_sql_id = (SELECT unique_sql_id FROM p)
);
              query               | n_soft_parse | n_hard_parse | n_returned_rows | db_time | cpu_time |
execution_time | parse_time | plan_time | rewrite_time | pl_execution_time |
pl_compilation_time | data_io_time
----------------------------------+--------------+--------------+-----------------+---------+----------+----------------
+------------+-----------+--------------+-------------------+-
--------------------+--------------
 call test_proc1();               |            0 |            2 |               1 |   21590 |    13130 |          13769 |        327
|       728 |          11 |            12539 |
               659 |         2348
 SELECT COUNT(*)          FROM test |            0 |            1 |               1 |   19194 |    17705 |            289
|        327 |       728 |          11 |             9324 |
               659 |         2348
(2 rows)

-- Similarly, query DBE_PERF.statement_history in the system database.
gaussdb=# WITH p AS (SELECT * FROM DBE_PERF.STATEMENT_HISTORY WHERE query LIKE 'call
test_proc1%' LIMIT 1)
SELECT
parent_unique_sql_id,
unique_query_id,
query,
n_soft_parse,
n_hard_parse,
n_returned_rows,
db_time,
cpu_time,
execution_time,
parse_time,
plan_time,
rewrite_time,
pl_execution_time,
pl_compilation_time,
data_io_time
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
FROM (
SELECT * FROM DBE_PERF.STATEMENT_HISTORY
WHERE unique_query_id = (SELECT unique_query_id FROM p)
UNION ALL
SELECT * FROM DBE_PERF.STATEMENT_HISTORY
WHERE parent_unique_sql_id = (SELECT unique_query_id FROM p)
);
 parent_unique_sql_id | unique_query_id |               query               | n_soft_parse | n_hard_parse |
n_returned_rows | db_time | cpu_time | execution_time | parse_time | plan_t
ime | rewrite_time | pl_execution_time | pl_compilation_time | data_io_time
---------------------+-----------------+-----------------------------------+--------------+--------------
+-----------------+---------+----------+----------------+------------+-------
----+--------------+-------------------+---------------------+--------------
                  0 |      4276558564 | call test_proc1();                |            0 |            2 |               1 |   14842
|     7871 |         13480 |          0 |
  0 |           0 |             12539 |                   0 |            0
          4276558564 |      1011671199 | SELECT COUNT(*)          FROM test |            0 |            1 |
1 |   12446 |    12446 |             0 |        327 |
728 |          11 |             9324 |                659 |          615
(2 rows)

-- Query dbe_perf.statement_history in non-system databases. No data is stored.
gaussdb=# \c tedb
tedb=# WITH p AS (SELECT * FROM DBE_PERF.STATEMENT_HISTORY WHERE query LIKE 'call test_proc1%'
LIMIT 1)
SELECT
parent_unique_sql_id,
unique_query_id,
query,
n_soft_parse,
n_hard_parse,
n_returned_rows,
db_time,
cpu_time,
execution_time,
parse_time,
plan_time,
rewrite_time,
pl_execution_time,
pl_compilation_time,
data_io_time
from (
SELECT * FROM DBE_PERF.STATEMENT_HISTORY
WHERE unique_query_id = (SELECT unique_query_id FROM p)
UNION ALL
SELECT * FROM DBE_PERF.STATEMENT_HISTORY
WHERE parent_unique_sql_id = (SELECT unique_query_id FROM p)
);
 parent_unique_sql_id | unique_query_id | query | n_soft_parse | n_hard_parse | n_returned_rows | db_time |
cpu_time | execution_time | parse_time | plan_time | rewrite_time | pl_execu
tion_time | pl_compilation_time | data_io_time
---------------------+-----------------+-------+--------------+--------------+-----------------+---------+----------
+----------------+------------+-----------+--------------+---------
----------+---------------------+--------------
(0 rows)

-- Switch back to the initial database and drop the test database.
tedb=# \c postgres
gaussdb=# DROP DATABASE tedb;
```

7.24.3 Typical Issues

Q: How do I analyze a slow stored procedure?

A: You can analyze the cause of the slow execution by referring to the following process:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If the stored procedure is being executed, you can query the execution status of the stored procedure through pg_stat_activity. After obtaining the session ID through pg_stat_activity, you can call the trace API to monitor the execution information and statement execution plan in real time and further analyze the cause of the slow execution.
- If the stored procedure has been executed and the call path is uncertain, you can use dbe_perf.statement_history to search for slow statements, analyze the execution plan and wait event of the corresponding statements, and find the outermost call by setting **debug_query_id** to the same value and **parent_unique_sql_id** to **0**.

Q: How do I analyze the execution of a stored procedure using pg_stat_activity and trace?

A: Assume that a session is executing a stored procedure for batch processing. The execution takes a long time and does not end. You can analyze the problem as follows:

1. Run the following statement to check the ID of the session that is executing the stored procedure:

```
select sessionid, query from pg_stat_activity where query like '%call%' and sessionid != pg_current_sessid();
```

2. Call the dbe_perf.pl_trace_func_start(sessionid) function to trace the corresponding session.
3. You can use the dbe_perf.pl_get_trace_info function to observe the trace information in real time and analyze the cause of the slow execution based on the statement execution plan.

7.25 Database Connection Driver Guide

7.25.1 JDBC

7.25.1.1 Overview

Java Database Connectivity (JDBC) provides a unified API for accessing various relational databases, enabling Java applications to interact with various databases to implement data persistence and management. JDBC allows Java applications to use SQL statements to perform database operations, such as querying, inserting, updating, and deleting data.

GaussDB supports JDBC 4.2. You can use JDK 1.8, JDK 17, and JDK 21 (the driver package of the corresponding version is required) to compile program code. JDBC-ODBC bridging is not supported.

7.25.1.2 Technical Principle

The JDBC architecture consists of the following components:

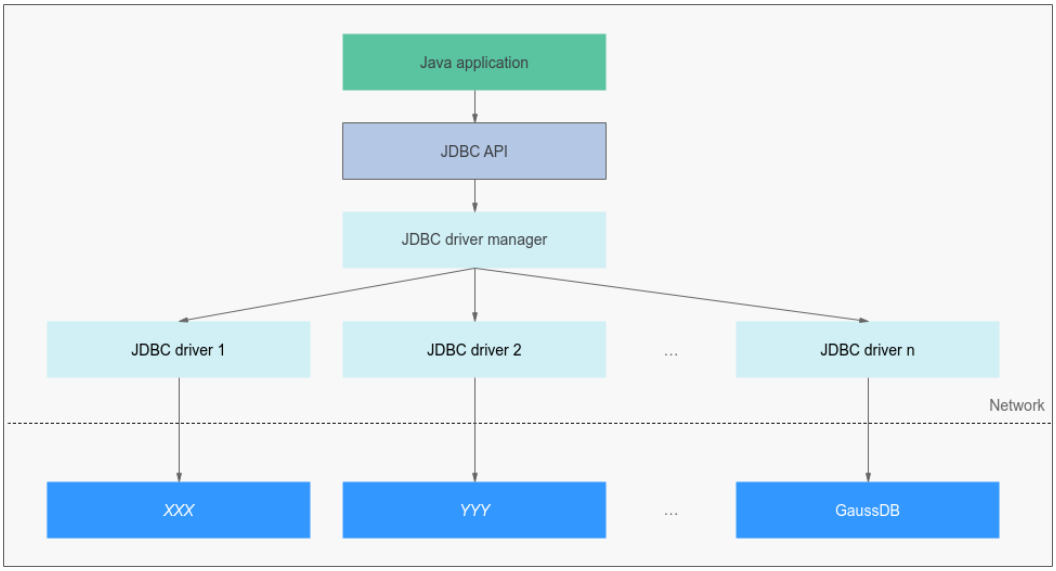
- JDBC APIs
- JDBC driver manager

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- JDBC drivers

A Java application initiates a database operation request (such as query and update) and communicates with the JDBC driver manager through JDBC APIs. The JDBC driver manager loads and manages JDBC drivers. The JDBC driver communicates with a specific database and sends SQL statements to the database. The database executes SQL statements and returns a result set. **Figure 7-40** shows the JDBC system architecture.

Figure 7-40 JDBC system architecture



API Description

JDBC APIs are provided for Java applications to interact with different databases. The java.sql package is the core JDBC API. **Table 7-29** describes some common APIs and methods in JDBC, covering common operations such as connecting to a database, executing SQL statements, obtaining result sets, and closing connections.

NOTE

For details about JDBC APIs and methods, see **JDBC API Reference**.

Table 7-29 Common JDBC APIs and methods

API	API Description	Key Method	Method Description
java.sql.DriverManager	Manages JDBC drivers and provides an entry for obtaining database connections.	getConnection(String url, String user, String password)	Establishes a database connection using the URL, username, and password.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	API Description	Key Method	Method Description
		getConnection(String url, Properties info)	Establishes a database connection using the URL and properties (including more parameters).
java.sql.Connection	Represents a session with the database, manages transactions, and creates statements.	createStatement()	Creates a statement.
		prepareStatement(String sql)	Creates a prepared statement.
		prepareCall(String sql)	Creates a callable statement.
		close()	Closes a database connection.
java.sql.Statement	Executes static SQL statements (without parameters).	executeQuery(String sql)	Executes query statements and returns result sets.
		executeUpdate(String sql)	Executes data modification statements (such as addition, deletion, and modification) and returns the number of affected rows.
		execute(String sql)	Executes any SQL statement and returns a Boolean value indicating whether there is a result set.
		addBatch(String sql)	Adds a batch processing statement.
		executeBatch()	Executes batch processing.
		close()	Closes a statement.
java.sql.PreparedStatement	Prepares an SQL statement (parameterized query is supported to prevent SQL injection).	setInt(int parameterIndex, int x)	Sets parameters.
		setString(int parameterIndex, String x)	
		setFloat(int parameterIndex, float x)	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	API Description	Key Method	Method Description
		executeQuery()	Executes a prepared query statement and returns a result set.
		executeUpdate()	Executes a prepared data modification statement and returns the number of affected rows.
		execute()	Executes any prepared statement and returns a Boolean value indicating whether there is a result set.
		addBatch()	Adds a prepared statement for batch processing.
		executeBatch()	Executes batch processing.
		close()	Closes a prepared statement.
java.sql.CallableStatement	Calls a database stored procedure.	registerOutParameter(int parameterIndex, int type)	Registers an output parameter.
		getInt(int columnIndex)	Obtains the output parameter value of a stored procedure.
		getString(int columnIndex)	
		getFloat(int columnIndex)	
java.sql.ResultSet	Queries a result set.	next()	Moves the cursor to the next row.
		getInt(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as an integer.
		getInt(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as an integer.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	API Description	Key Method	Method Description
		getString(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as a string.
		getString(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as a string.
		getMetaData()	Retrieves result set metadata (such as column names and types).
		close()	Closes a result set.
java.sql.ResultSetMetaData	Describes result set metadata (such as column names and types).	getColumnCount()	Retrieves the number of columns.
		getColumnName(int column)	Retrieves the column name.
		getColumnTypeName(int column)	Retrieves the column type.

7.25.1.3 Operation Guide

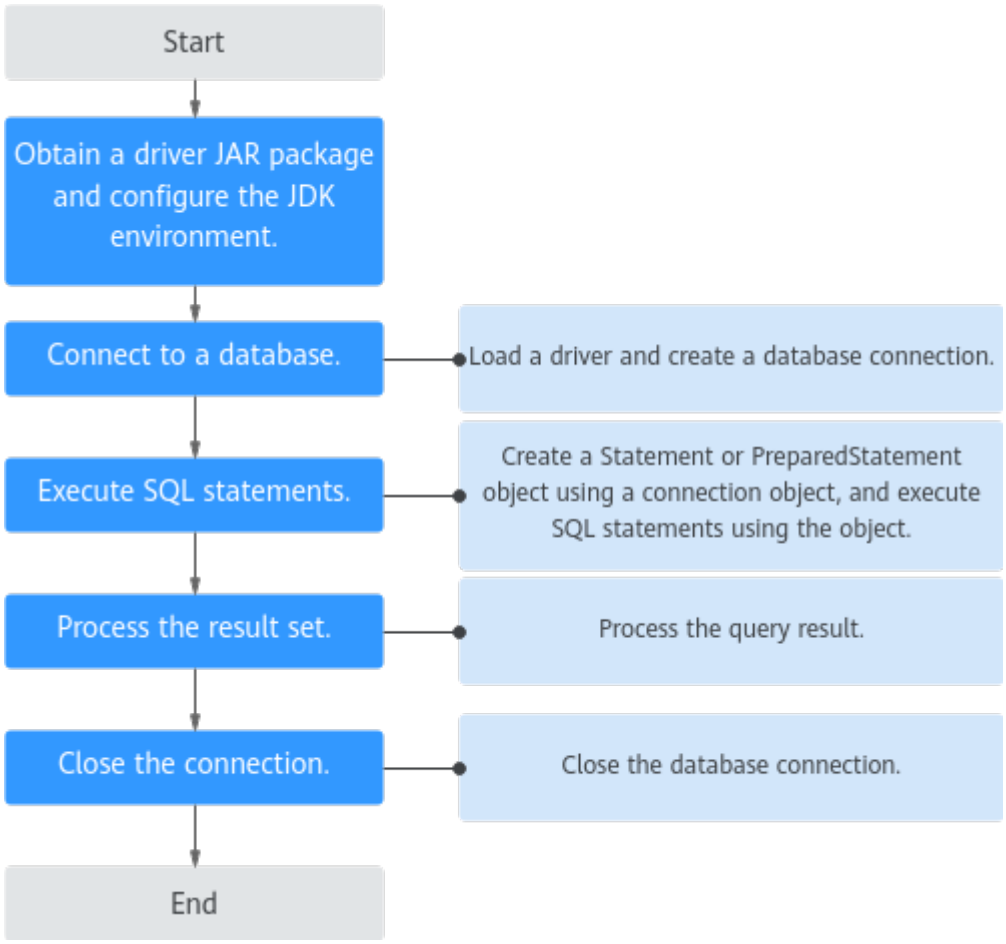
7.25.1.3.1 Development Process

Figure 7-41 shows the application development process based on JDBC.

This process involves these steps: obtain the driver JAR package and configure the JDK environment, connect to a database, execute SQL statements, process the result set, and close the connection.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-41 Application development process based on JDBC



7.25.1.3.2 Obtaining the Driver JAR Package and Configuring the JDK Environment

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

Obtaining the Driver JAR Package

Decompress the release package to obtain the corresponding driver JAR packages. The JDK 8 release package provides the driver packages that support the JDK 1.8 environments. The JDK 17 release package provides the driver package that supports the JDK 17 environment. The JDK 21 release package provides the driver package that supports the JDK 21 environment.

The directory structure of the JDK 8 release package (x86 is used as an example) is as follows:

- **GaussDB-Kernel**_*Database version number*_**Jdbc_X86_Jdk8_Distributed.tar.gz**
  - **GaussDB-Kernel**_*Database version*_*OS version*_**64bit_Jdbc.tar.gz**
    - gaussdbjdbc.jar
    - gscejdbc.jar

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

        - gsjdbc4.jar
        - gsjdbc200.jar
        - opengaussjdbc.jar
        - opengaussjdbc-ce.jar

    The directory structure of the JDK 17 release package (x86 is used as an example) is as follows:

- **GaussDB-Kernel**_*Database version*_**Jdbc_X86_Jdk17_Distributed.tar.gz**
  - **GaussDB-Kernel**_*Database version_OS version*_**64bit_Jdbc.tar.gz**
    - gaussdbjdbc.jar
    - gscejdbc.jar
    - gsjdbc4.jar
    - gsjdbc200.jar
    - opengaussjdbc.jar
    - opengaussjdbc-ce.jar

    The directory structure of the JDK 21 release package (x86 is used as an example) is as follows:

- **GaussDB-Kernel**_*Database version*_**Jdbc_X86_Jdk21_Distributed.tar.gz**
  - **GaussDB-Kernel**_*Database version_OS version*_**64bit_Jdbc.tar.gz**
    - gaussdbjdbc.jar
    - gscejdbc.jar
    - gsjdbc4.jar
    - gsjdbc200.jar
    - opengaussjdbc.jar
    - opengaussjdbc-ce.jar

    Obtain the corresponding JAR driver package from the release package as required.

- For the JDK 1.8 environment, use the JDK 8 release package.
  - **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This package applies to the JDK 1.8 environment.
  - **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependency libraries related to encryption and decryption that need to be loaded to the encrypted database. This driver

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

package applies to encrypted scenarios. Currently, only EulerOS is supported.

- For the JDK 17 environment, use the JDK 17 release package.
  - **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This package applies to the JDK 17 environment.
  - **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependency libraries related to encryption and decryption that the encrypted database requires. It applies to encrypted scenarios. Currently, only EulerOS is supported.
- For the JDK 21 environment, use the JDK 21 release package.
  - **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This package applies to the JDK 21 environment.
  - **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependency libraries related to encryption and decryption that the encrypted database requires. It applies to encrypted scenarios. Currently, only EulerOS is supported.

NOTICE

- Before using the driver package **gscejdbc.jar**, you need to set the environment variable *LD_LIBRARY_PATH*. For details, see **Using JDBC to Operate on an Encrypted Database**.
- The JAR packages of the JDBC release package are classified by architecture. The **gscejdbc.jar** package can be used only when it is consistent with the deployment environment. Other JAR packages do not need to be consistent with the deployment environment.
- The drivers of the JDK 17 and JDK 21 release packages do not support the new features of JDBC 4.3.

Configuring the JDK Environment

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

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

```
java -version
```

Check whether JDK 1.8 has been installed. If not, download the installation package from the **official website** and install it.

Step 2 Configure system environment variables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1. Right-click **My computer** and choose **Properties**.
2. In the **System** window, click **Advanced system settings** in the navigation pane.

   NOTE

   The operation method may vary depending on the OS version.
3. In the **System Properties** dialog box, click **Environment Variables** on the **Advanced** tab page.
4. In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see **Table 7-30**.

Table 7-30 Variables

Variable	Operation	Variable Value
JAVA_HOME	– If the variable exists, click **Edit**. – If the variable does not exist, click **New**.	Specifies the Java installation directory. Example: **C:\Program Files\Java\jdk1.8.0_131**
Path	Click **Edit**.	– If *JAVA_HOME* is configured, add *%JAVA_HOME%***\bin** before the variable value. – If *JAVA_HOME* is not configured, add the full Java installation path before the variable value: **C:\Program Files\Java\jdk1.8.0_131\bin**
CLASSPATH	Click **New**.	*%JAVA_HOME%***\lib**; *%JAVA_HOME%***\lib\tools.jar**

----End

7.25.1.3.3 Creating a Database Connection

You can call the DriverManager.getConnection method to create a database connection only after the driver is loaded. This section describes how to create a database connection using the DriverManager.getConnection(String url, String user, String password) and DriverManager.getConnection(String url, Properties info) methods.

getConnection(String url, String user, String password)

To create a database connection using the DriverManager.getConnection(String url, String user, String password) method, perform the following steps:

Step 1 Import java.sql.Connection and java.sql.DriverManager.

```
import java.sql.Connection;
import java.sql.DriverManager;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

java.sql.Connection is a database connection API. The getConnection() method of java.sql.DriverManager is used to connect applications to a database. You may also import other APIs and classes as needed. For details, see **JDBC API Reference**.

Step 2 Specify the database **sourceURL** (change *$ip*, *$port*, and *database* as required), username, and password.

Writing the username and password to code might bring security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:gaussdb://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Load the driver.

1. Add the **gaussdbjdbc.jar** package to the runtime environment (for example, the IDE).
2. Load the database driver **com.huawei.gaussdb.jdbc.Driver**.

```
String driver = "com.huawei.gaussdb.jdbc.Driver";
Class.forName(driver);
```

Step 4 Create a database connection.

Call DriverManager.getConnection(String url, String user, String password) to connect to the database.

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

----End

getConnection(String url, Properties info)

To create a database connection using the DriverManager.getConnection(String url, Properties info) API, perform the following steps:

Step 1 Import java.sql.Connection, java.sql.DriverManager, and java.util.Properties.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

The setProperty() method of java.util.Properties is used to set properties. You may also import other APIs and classes as needed. For details, see **JDBC API Reference**.

Step 2 Specify the database **sourceURL** (change *$ip*, *$port*, and *database* as required), username, and password.

Writing the username and password to code might bring security risks. You are advised to store the username and password in environment variables.

```
String sourceURL = "jdbc:gaussdb://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

Step 3 Create a Properties object and set **userName** and **password** as the values of the object.

```
Properties info = new Properties();
info.setProperty("user", userName);
info.setProperty("password", password);
```

Step 4 Load the driver.

1. Add the **gaussdbjdbc.jar** package to the runtime environment (for example, the IDE).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. Load the database driver **com.huawei.gaussdb.jdbc.Driver**.

```
String driver = "com.huawei.gaussdb.jdbc.Driver";
Class.forName(driver);
```

Step 5 Create a database connection.

Call DriverManager.getConnection(String url, Properties info) to connect to the database.

```
Connection conn = DriverManager.getConnection(sourceURL, info);
```

----End

7.25.1.3.4 Executing SQL Statements

Applications operate a database by executing SQL statements. This section uses examples to describe how to perform operations such as CREATE TABLE, INSERT, UPDATE, and SELECT.

- To create a table, refer to **Executing a Common SQL Statement**.
- To insert data in batches, refer to **Executing a Batch INSERT Statement**.
- To update data, refer to **Executing a Prepared Statement for Update**.
- To return the result set, refer to **Executing a Query Statement**.

Executing a Common SQL Statement

Prerequisites: You have connected to the database using the connection object conn.

Create the **customer_t1** table as follows:

Step 1 Create a statement object by calling the createStatement() method of the Connection API.

```
Statement stmt = conn.createStatement();
```

Step 2 Run the SQL statement by calling the executeUpdate() method of the Statement API.

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

Step 3 Close the statement object stmt by calling the close() method of the Statement API.

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

----End

Executing a Batch INSERT Statement

A prepared statement can be created for batch insertion. The database creates an execution plan only once, saving the statement compilation and optimization time.

Prerequisites: The **customer_t1** table has been created by referring to **Executing a Common SQL Statement**.

Execute the prepared statement to insert data in batches as follows:

Step 1 Create the prepared statement object **pst** by calling the prepareStatement() method of the Connection API.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
PreparedStatement pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
```

Step 2 Call the corresponding API to set parameters for each piece of data and call addBatch() to add SQL statements to the batch processing.

```
for (int i = 0; i < 3; i++) {
  pst.setInt(1, i);
  pst.setString(2, "data " + i);
  pst.addBatch();
}
```

Step 3 Perform batch insertion by calling the executeBatch() method of the PreparedStatement API.

```
pst.executeBatch();
```

Step 4 Close the prepared statement object **pst** by calling the close() method of the PreparedStatement API.

```
pst.close();
```

----End

NOTE

During batch processing, you must adhere to the following key principles to ensure database performance:

1. Continuous execution
   - The batch processing program must run continuously without interruption.
   - An abnormal stop significantly impacts overall database performance.
2. Optimized transaction management
   - Disable autocommit.
     ```
     conn.setAutoCommit(false);
     ```
   - Commit in batches rather than line by line.
   - Determine an appropriate batch size based on data volume and system resources.

Executing a Prepared Statement for Update

Prepared statements are complied and optimized once but can be used in different scenarios by assigning different values. This preparation helps reduce execution time. If you want to run a statement for several times, use a prepared statement.

Prerequisites: Data has been inserted into the **customer_t1** table by referring to **Executing a Batch INSERT Statement**.

To update data by executing a prepared SQL statement, perform the following steps:

Step 1 Create the prepared statement object **pstmt** by calling the prepareStatement() method of the Connection API.

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

Step 2 Set parameters by calling the setString() method of the PreparedStatement API.

```
pstmt.setString(1, "new Data");
```

Step 3 Execute the prepared statement by calling the executeUpdate() method of the PreparedStatement API.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 4 Close the prepared statement object **pstmt** by calling the close() method of the PreparedStatement API.

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

----End

NOTICE

After bound parameters are set in prepareStatement, a B packet or U packet is constructed and sent to the server when the SQL statement is executed. The maximum length of a B packet or U packet cannot exceed 1023 MB. If the data bound at a time is too large, an exception may occur because the packet is too long. Therefore, when setting bound parameters in prepareStatement, you need to estimate and control the size of the bound data to avoid exceeding the upper packet limit.

Executing a Query Statement

Prerequisites: You have performed operations in **Executing a Prepared Statement for Update**.

To execute a query statement, perform the following steps:

Step 1 Create the statement object **stmts** by calling the createStatement() method of the Connection API.

```
Statement stmts = conn.createStatement();
```

Step 2 Call the executeQuery() method of the Statement API to execute a query statement and return a result set.

```
ResultSet rs = stmts.executeQuery("SELECT * FROM customer_t1");
```

----End

7.25.1.3.5 Processing the Result Set

After **executing a query statement**, you need to process the returned result set.

Step 1 Use the while loop and next() method to traverse the result set row by row.

```
while (rs.next()) {
    // Obtain values by column index.
    int c_customer_sk = rs.getInt(1);       // Obtain int data in the first column.
    String c_customer_name = rs.getString(2);      // Obtain string data in the second column.
    System.out.println("c_customer_sk: " + c_customer_sk + ", c_customer_name: " + c_customer_name);
}
```

Step 2 Call the close() method to close resources.

```
rs.close();    // Close the ResultSet object.
stmts.close();    // Close the Statement object.
```

----End

7.25.1.3.6 Closing a Database Connection

After completing required data operations in a database through a database connection, call the close() method to close the database connection.

```
conn.close();
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> The Connection, Statement, and ResultSet APIs all provide a close() method, which should be used to close objects after use.

7.25.1.3.7 Best Practices

GaussDB provides best practices for JDBC.

For details, see "Best Practices for JDBC" in *Best Practices*.

7.25.1.4 Typical Issues

7.25.1.4.1 Incorrect batchMode Settings

Symptom

In the scenario where the URL parameters **batchMode** is set to **on** and **reWriteBatchedInserts** is set to **true**, after JDBC is used to insert data in batches, a message is displayed, indicating that the number of bound parameters is inconsistent with the number of parameters required by the statement.

```
bind message supplies * parameters, but prepared statement "" requires *
```

Example:

```
// conn is a created connection object. The URL parameters for creating the connection contain batchMode=on&reWriteBatchedInserts=true.
// Bind parameters in batches and then execute the statement. The number of bound parameters does not match the number of columns in the rewritten INSERT statement. As a result, an exception is reported.
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6

PreparedStatement stmt = conn.prepareStatement("INSERT INTO test_tbl VALUES (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.executeBatch();
```

Cause Analysis

When **reWriteBatchedInserts** is set to **true**, multiple SQL statements are combined into one. As a result, the number of reserved parameter columns in the statement changes. If **batchMode** is set to **on**, parameters are bound based on the SQL statements before combination. As a result, the number of bound parameters is inconsistent with the number of parameters required by the statement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Solution

If **reWriteBatchedInserts** is set to **true**, set **batchMode** to **off**.

7.25.1.4.2 An Error Is Reported When Validation Is Enabled for Data Insertion in the Hibernate Framework

Symptom

The customer migrates data from an ORA-compatible database to GaussDB, and uses the DRS tool to migrate table definitions. After the migration, the original service code of the customer is unavailable, and an error is reported when the Hibernate framework validates the table definitions.

```
Schema-validation: wrong column type encountered in column [execute_time] in table [abnormal_event]; found [int8 (Types#BIGINT)], but expecting [int4 (Types#INTEGER)]
```

Examples:

- The structure of the **Student** table in the ORA-compatible database is as follows:
  ```
  id number(15,0)
  sid number(15,0)
  age number(10,0)
  ```
- After data is migrated to GaussDB using DRS, the **Student** table definition is as follows:
  ```
  id bigint
  sid bigint
  age Integer
  ```
- When the **Student** entity class information in the actual database is as follows:
  ```
  Long id
  Long sid
  Long age  -- An error is reported when the Hibernate framework validates the table definition.
  ```

Configuration file **hibernate.cfg.xml**:

```
<hibernate-configuration>
    <session-factory>
        <!-- GaussDB connection information -->
        <property name="connection.driver_class">com.huawei.gaussdb.jdbc.Driver</property>
        <property name="connection.url">jdbc:gaussdb://x.x.x.x:xx/postgres?currentSchema=public</property>
        <property name="connection.username">xxxxxx</property>
        <property name="connection.password">xxxxxx</property>

        <!-- The following configurations are optional: -->

        <!-- Specify whether to support dialects. -->
        <!-- In PG-compatible mode, the following configuration must exist: -->
         <property name="dialect">org.hibernate.dialect.PostgreSQL92Dialect</property>
        <!-- In ORA-compatible mode, the following configuration is recommended: -->
<!--        <property name="dialect">org.hibernate.dialect.OracleDialect</property>-->

        <property name="met"></property>

        <!-- Specify whether to print SQL statements during CURD. -->
        <property name="show_sql">true</property>
        <!-- The table is created automatically. -->
<!--        <property name="hbm2ddl.auto">create</property>-->
        <!-- Enable the validation when inserting data. -->
        <property name="hbm2ddl.auto">validate</property>
        <!-- Disable the validation. -->
<!--        <property name="hbm2ddl.auto">none</property>-->
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        <!-- Register resources (entity class mapping file). -->
        <mapping resource="student.xml"/>
    </session-factory>
</hibernate-configuration>
```

Main functions for inserting data:

```
// Create the object to be tested.
Student student = new Student();
student.setId(20L);
student.setAge(222L);
student.setSid(222L);

// Start a transaction and obtain an object based on the session.
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

// Save data using a session.
session.save(student);

// Commit the transaction.
transaction.commit();

// After the operation is complete, close the session.
session.close();
```

Cause Analysis

1. Reason why data is successfully inserted into an ORA-compatible database and passes verification: When verifying data, the Hibernate framework compares **TypeCode** of the inserted data type java.lang.Long with **TypeCode** of the number type in the table definition. If they are inconsistent, Hibernate then compares **sqlType** of the inserted long type with **Type** of the table definition and converts the long type into **number(19,0)** (this mapping is maintained by org.hibernate.dialect.OracleDialect). Because the table definition uses the number type and **number(19,0)** also begins with **number**, the verification succeeds. When working with an ORA-compatible database, you can use the java.lang.Long type to insert any integer data involved in service code.
2. The validation in GaussDB is different and data can be inserted successfully after the validation is disabled. When the Hibernate framework validates the inserted data, no error is reported because the service code is of the java.lang.Long type, which corresponds to **TypeCode** of the table data type bigint. However, if the table data type is integer, the system checks **TypeCode** of the java.lang.Long type in the service code and compares it with that of the integer type in the table data. If they are inconsistent, the system converts the long type to int8 based on the org.hibernate.dialect.PostgreSQLDialect mapping. If the table data type integer is converted to int4, which does not match, the validation will fail. When GaussDB is used, you must use the Java integer corresponding to the table type when inserting data of the integer type.

Solution

To rectify the fault, do as follows:

- Disable the validation functionality.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Modify the service code as authorized by the customer and use the corresponding Java integer for different table data types.

7.25.1.4.3 An Error Is Reported or A Connection Is Blocked in SSL Mode

Symptom

When a JDBC connection is established in SSL mode, a strong random number is generated on the client. During connection establishment, errors may occur.

Scenario 1: The error shown in the following figure is reported.

```
"Thread-0" #18 prio=5 os_prio=0 tid=0x00007f2ad0385000 nid=0x5429 runnable [0x00007f2aa069b000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileInputStream.readBytes(Native Method)
        at java.io.FileInputStream.read(FileInputStream.java:255)
        at sun.security.provider.NativePRNG$RandomIO.readFully(NativePRNG.java:424)
        at sun.security.provider.NativePRNG$RandomIO.ensureBufferValid(NativePRNG.java:526)
        at sun.security.provider.NativePRNG$RandomIO.implNextBytes(NativePRNG.java:545)
        - locked <0x000000067273a950> (a java.lang.Object)
        at sun.security.provider.NativePRNG$RandomIO.access$400(NativePRNG.java:331)
        at sun.security.provider.NativePRNG$Blocking.engineNextBytes(NativePRNG.java:268)
        at java.security.SecureRandom.nextBytes(SecureRandom.java:468)
        at java.security.SecureRandom.next(SecureRandom.java:491)
        at java.util.Random.nextInt(Random.java:329)
        at sun.security.ssl.SSLContextImpl.engineInit(SSLContextImpl.java:106)
        at javax.net.ssl.SSLContext.init(SSLContext.java:282)
        at org.postgresql.ssl.LibPQFactory.<init>(LibPQFactory.java:175)
        at org.postgresql.core.SocketFactoryFactory.getSslSocketFactory(SocketFactoryFactory.java:62)
        at org.postgresql.ssl.MakeSSL.convert(MakeSSL.java:33)
        at org.postgresql.core.v3.ConnectionFactoryImpl.enableSSL(ConnectionFactoryImpl.java:723)
        at org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect(ConnectionFactoryImpl.java:203)
        at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:330)
        at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:58)
        at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:357)
```

Scenario 2: The connection is blocked. If **loginTimeout** is set in the URL connection string, the message "Connection attempt timed out" is displayed. If this parameter is not set, the connection is always blocked.

Cause Analysis

The entropy source is insufficient and the random number generation on the client is too slow to meet product requirements. As a result, the service fails to be started. This problem exists in some Linux environments.

Solution

Solution 1: Start the haveged service on the client and increase the entropy value of the system entropy pool to improve the speed of reading random numbers. The startup command is as follows:

```
systemctl start haveged
```

Method 2: Adjust the JDK configuration on the client.

Open the *$JAVA_PATH*/**jre/lib/security/java.security** file and modify the configuration items as follows:

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

7.25.2 ODBC

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.25.2.1 Overview

Open Database Connectivity (ODBC) is a database access API proposed by Microsoft based on the X/OPEN CLI. It is implemented using the C/C++ language. ODBC provides a standard data access API, enabling applications to access heterogeneous database management systems (DBMSs) in a unified manner. This decouples database access from DBMS types and OS platforms.

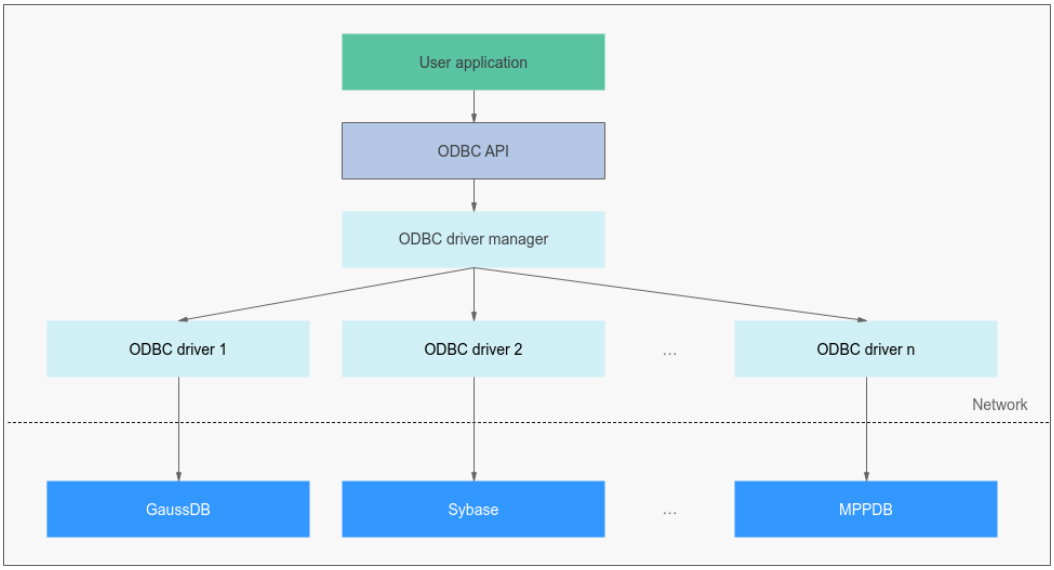
ODBC allows applications to use SQL statements to query, insert, update, and delete data in a database. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.

7.25.2.2 Technical Principle

The ODBC architecture consists of three main components: application, ODBC driver manager, and ODBC driver. Applications use ODBC APIs to communicate with the ODBC driver manager which loads and manages ODBC drivers. The ODBC drivers are responsible for communicating with a specific database, executing SQL queries, and returning results. The ODBC architecture is classified into two types based on the OS type (Unix/Linux or Windows). The overall architecture is the same. Applications call a database driver through the driver manager to connect to the database. The difference lies in the driver manager and the corresponding database driver.

For details about the ODBC system architecture, see **Figure 7-42**.

Figure 7-42 ODBC system architecture



Platforms Supported by ODBC

GaussDB supports ODBC 3.5 in the following environments.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-31 Platforms supported by ODBC

OS	Platform
EulerOS V2.0SP5	x86 64-bit
EulerOS V2.0SP9	Arm 64-bit
EulerOS V2.0SP10	x86 64-bit
EulerOS V2.0SP10	Arm 64-bit
Windows 7	x86 32-bit
Windows 7	x86 64-bit
Windows Server 2008	x86 32-bit
Windows Server 2008	x86 64-bit
Kylin V10	x86 64-bit
Kylin V10	Arm 64-bit
UnionTech V20	x86 64-bit
UnionTech V20	Arm 64-bit
Huawei Cloud EulerOS 2.0	x86 64-bit
Huawei Cloud EulerOS 2.0	Arm 64-bit
Suse 12.5	x86 64-bit
openEuler 22.03 LTS SP4	Arm 64-bit
BCLinux-R8-U2	x86 64-bit
BCLINUX21.10	x86 64-bit
REDHAT 8.10	x86 64-bit
REDHAT 8.10	Arm 64-bit
NFSChina Server 4.0	x86 64-bit
NFSChina Server 4.0	Arm 64-bit
Ubuntu 22.04 LTS	x86 64-bit
Ubuntu 22.04 LTS	Arm 64-bit
Suse15.7	x86 64-bit
Suse15.7	Arm 64-bit
Huawei Cloud EulerOS 3.0	x86 64-bit
Huawei Cloud EulerOS 3.0	Arm 64-bit
Kylin V11	Arm 64-bit

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The ODBC driver manager running on Unix or Linux can be unixODBC or iODBC. Here, unixODBC 2.3.7 is used as the component for connecting to a database.

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

NOTE

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

By default, there is no libltdl library on SUSE 12 SP5. You need to install the libltdl library before using the ODBC.

API Description

ODBC provides a set of standard C language APIs, enabling applications to communicate with different types of databases. ODBC APIs are widely used for interaction between programming languages and DBMSs. **Table 7-32** describes some common ODBC APIs (functions), covering common operations such as connecting to databases, executing queries, and obtaining results.

NOTE

For details about the APIs, see **ODBC API Reference**.

Table 7-32 Common ODBC APIs

API	Description	Prototype
SQLAllocHandle	Allocates handles for environments, connections, statements, or descriptors.	SQLRETURN SQLAllocHandle(SQLSMALLINT HandleType, SQLHANDLE InputHandle, SQLHANDLE *OutputHandlePtr);
SQLConnect	Establishes a connection between a driver and a data source. After the connection is established, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.	SQLRETURN SQLConnect(SQLHDBC ConnectionHandle, SQLCHAR *ServerName, SQLSMALLINT NameLength1, SQLCHAR *UserName, SQLSMALLINT NameLength2, SQLCHAR *Authentication, SQLSMALLINT NameLength3);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Prototype
SQLPrepare	Prepares an SQL statement to be executed. Note that ODBC does not support the kernel reuse plan when sending prepared statements and a new plan needs to be generated for each execution, causing high CPU usage. If plan reuse is required, you are advised to enable the **MaxCacheQueries** parameter.	SQLRETURN SQLPrepare(SQLHSTMT StatementHandle, SQLCHAR *StatementText, SQLINTEGER TextLength);
SQLExecDirect	Executes a prepared statement using the current handle. SQLExecDirect is the fastest method for executing only one SQL statement at a time.	SQLRETURN SQLExecDirect(SQLHSTMT StatementHandle, SQLCHAR *StatementText, SQLINTEGER TextLength);
SQLExecute	Executes an SQL statement that has been prepared by using the SQLPrepare function.	SQLRETURN SQLExecute(SQLHSTMT StatementHandle);
SQLGetData	SQLGetData is used to retrieve data for a single column in the result set. It can be called multiple times to partially retrieve variable-length data.	SQLRETURN SQLGetData(SQLHSTMT StatementHandle, SQLUSMALLINT Col_or_Param_Num, SQLSMALLINT TargetType, SQLPOINTER TargetValuePtr, SQLLEN BufferLength, SQLLEN *StrLen_or_IndPtr);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API	Description	Prototype
SQLFetch	Moves the cursor to the next row of the result set and retrieves any bound columns.	SQLRETURN SQLFetch(SQLHSTMT StatementHandle);
SQLDisconnect	Closes a database connection handle.	SQLRETURN SQLDisconnect(SQLHDBC ConnectionHandle);

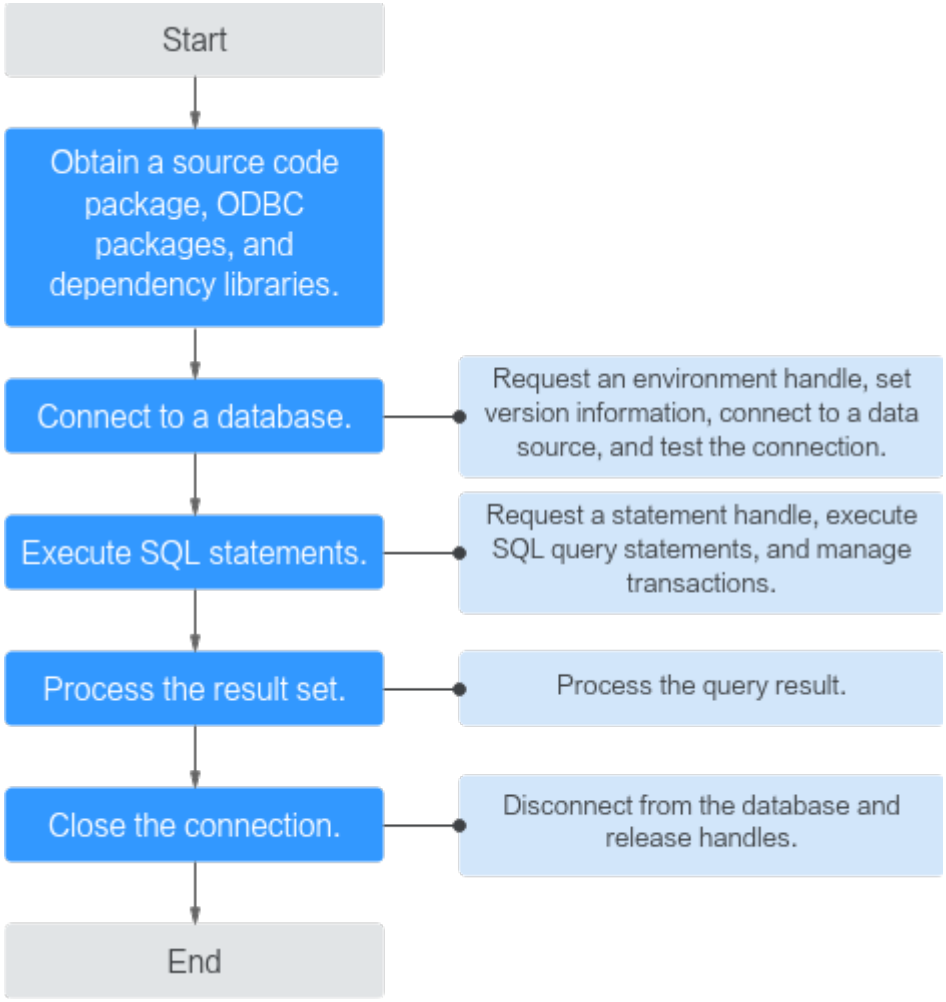
7.25.2.3 Operation Guide

7.25.2.3.1 Development Process

Figure 7-43 shows the development process based on ODBC.

This process involves five steps: obtain the source code package, ODBC package, and dependency library, connect to a database, execute SQL statements, process the result set, and close the connection.

Figure 7-43 Development process based on ODBC



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.25.2.3.2 Obtaining the Source Code Package, ODBC Package, and Dependency Library

To develop applications based on ODBC, you need to obtain the unixODBC source code package, ODBC package, and dependency library.

Obtaining the Source Package

- To obtain the unixODBC source code package, visit **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**.
- To obtain the MD5 file, visit **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**.

Use the MD5 file to verify the integrity of the unixODBC source code package. (Check whether the MD5 value is the same as that in the source code package.)

Obtaining the Driver Package

Download the driver package and its verification package listed in **Table 7-33**.

Table 7-33 Driver package download list

Version	Download Address
V2.0-10.x	**Driver package** **Verification package for the driver package**

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

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

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

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

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

     certutil -hashfile *{local_directory_of_the_driver_package}**{driver_package_name}* **sha256**

     - Replace *{local_directory_of_the_driver_package}* with the actual download path, for example, **C:\Users**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Replace *{driver_package_name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.

Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**

c. Compare the hash value obtained in **b** with the hash value of the verification package obtained in **Table 7-33**.

- If they are consistent, the verification is successful.
- If they are inconsistent, download the driver package again and repeat **a** to **c** to verify the driver package.

Obtaining the ODBC Package and Dependency Library

The ODBC package and dependency library are specific to Linux and Windows OSs. Obtain the appropriate ones based on your OS.

- Linux OS:

  Obtain **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Odbc.tar.gz** from the driver package. When developing applications in Linux environments, you will need unixODBC header files such as **sql.h** and **sqlext.h**, as well as the **libodbc.so** library.

  The header files and the library can be obtained from the unixODBC-2.3.7 source code package.

- Windows OS:

  Obtain **GaussDB-Kernel**_*Database version number*_**Windows_X86_Odbc.tar.gz** (32-bit) or **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** (64-bit) from the driver package.

  In Windows environments, the header files and libraries required for application development are readily available within the OS, eliminating the need to obtain them manually.

7.25.2.3.3 Creating a Database Connection

Before using ODBC to connect to a database, you need to prepare the required resources. To connect to a database, you need to configure the ODBC data source or use the ODBC APIs or the driver to implement the communication and interaction between an application and a database. This section describes how to configure a data source and connect to a database in the Linux/Windows OS.

Configuring a Data Source in the Linux OS

Step 1 Install unixODBC. (By default, the unixODBC source code package has been obtained during environment preparation.) It does not matter if unixODBC of another version has been installed.

For example, to install unixODBC-2.3.7, run the commands below on the client.

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no # To perform compilation on an Arm server, add the configure parameter --build=aarch64-unknown-linux-gnu.
make
# The installation may require root permissions.
make install
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- unixODBC is installed in the **/usr/local** directory by default. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to achieve higher performance. However, this option may cause an application that passes an invalid handle to the ODBC API to fail instead of returning an SQL_INVALID_HANDLE error.

Step 2 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Odbc.tar.gz**. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 3 Configure a data source.

1. Configure the ODBC driver file.

   Add the following content to the **/usr/local/etc/odbcinst.ini** file:

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/gsqlodbcw.so
   setup=/usr/local/lib/gsqlodbcw.so
   ```

   Table 7-34 describes the parameters in the **odbcinst.ini** file.

   Table 7-34 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to Driver in DSN.	[GaussMPP]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/gsqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/gsqlodbcw.so

2. Configure the data source file.

   Add the following content to the **/usr/local/etc/odbc.ini** file:

   ```
   [gaussdb]
   Driver=GaussMPP
   # IP address of the database server
   Servername=127.0.0.1
   # Database name
   Database=db1
   # Database username
   Username = omm
   # Database user password
   Password=
   # Database listening port
   Port = 8000
   Sslmode = allow
   ```

   Table 7-35 describes the parameters in the **odbc.ini** file. For details about the parameters, see **Configuring a Data Source on Linux**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-35 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name.	[gaussdb]
Driver	Driver name, corresponding to DriverName in **odbcinst.ini**.	Driver=Gauss MPP
Servername	Server IP address. Multiple IP addresses can be configured. Both IPv4 and IPv6 are supported.	Servername=127.0.0.1
Database	Name of the database to connect.	Database=db1
Username	Database username.	Username = omm
Password	Database user password. **NOTE** – The ODBC driver clears in-memory passwords to ensure that user credentials are not retained in memory post-connection. – However, configuring this parameter may cause passwords to remain in memory for extended periods due to UnixODBC's caching of data source files. – When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored. – The password in the configuration file must comply with the following HTTP rules: ■ Characters must comply with the URL encoding specifications. For example, the exclamation mark (!) must be written as %21, and the percent sign (%) must be written as %25. Therefore, pay attention to the characters. ■ A plus sign (+) will be replaced by a space.	Password=**********
Port	Port number of the server. When load balancing is enabled, multiple port numbers can be configured, each corresponding to an IP address. If only one port number is configured in the presence of multiple IP addresses under load balancing, all the IP addresses will default to sharing the same port number—that is, the one specified.	Port = 8000

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
Sslmode	Specifies whether to enable the SSL connection. **NOTE** **Table 3 Sslmode options** describes the valid values of **Sslmode**.	Sslmode = allow
Debug	Specifies whether to enable the debug mode and, if enabled, the log output level. Value range: 0 to *INT_MAX*. – **0**: The debug mode is disabled. – > 0: The debug mode is enabled. Mylogs will be output for the gsqlODBC driver and stored in the **/tmp/** directory with file names like **mylog**_*xxx*.**log**. – **1**: Only FATAL logs are output. – **2**: FATAL and ERROR logs are output. – **3**: FATAL, ERROR, and WARN logs are output. – **4**: FATAL, ERROR, WARN, and INFO logs are output. – ≥ 5: Logs at all levels are output, including TRACE logs. The default value is **0**. **NOTE** 1. FATAL: reports fatal errors. 2. ERROR: reports exceptions. 3. WARN: reports warnings. 4. INFO: records key information during program runtime. 5. TRACE: provides debugging information for developers.	Debug=5

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
CommLog	Specifies whether to enable backend communication logging and, if enabled, the log output level. Value range: 0 to *INT_MAX*. – **0**: Backend communication logging is disabled. – > 0: Backend communication logging is enabled. Qlogs will be output for the gsqlODBC driver and stored in the **/tmp/** directory with file names like **gsqlodbc**_*xxx*.**log**. – **1**: Only ERROR logs are output. – **2**: ERROR and INFO logs are output. – ≥ 3: ERROR, INFO, and TRACE logs are output. The default value is **0**. **NOTE** 1. ERROR: reports exceptions. 2. INFO: records key information during program runtime. 3. TRACE: provides debugging information for developers.	CommLog=1
UseServerSidePrepare	Specifies whether to enable the extended query protocol for the database. Value range: **0** and **1** – The value **0** means disabled. – The value **1** means enabled. The default value is **1**.	UseServerSidePrepare=1
UseBatchProtocol	Specifies whether to enable the batch query protocol. If it is enabled, DML performance can be improved. Value range: **0** and **1** – If this parameter is set to **0**, the batch query protocol is disabled (mainly for communication with earlier database versions). – If this parameter is set to **1** and **support_batch_bind** is set to **on**, the batch query protocol is enabled. The default value is **1**.	UseBatchProtocol=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
ForExtensionConnector	Specifies whether the savepoint is sent. Value range: **0** and **1** – **0**: The savepoint is sent. – **1**: The savepoint is not sent. The default value is **1**.	ForExtensionConnector=1
ConnectionExtraInfo	Specifies whether to display the driver deployment path and process owner in the **connection_info** GUC parameter. Value range: **0** and **1** – **0**: disabled. – **1**: enabled. The default value is **0**. **NOTE** If this parameter is set to **1**, the ODBC driver reports the driver deployment path and process owner to the database.	ConnectionExtraInfo=1
BoolsAsChar	Specifies whether to process Boolean values as characters. Value range: **0** and **1** – If this parameter is set to **0**, the Boolean values are mapped to the SQL_BIT type. – If this parameter is set to **1**, the Boolean values are mapped to the SQL_CHAR type. The default value is **1**.	BoolsAsChar = 1
RowVersioning	Specifies whether to allow the application to check whether a row of data is modified by another user when the row of data is updated. Value range: **0** and **1** – **0**: Application detection is not allowed. – **1**: Application detection is allowed. The default value is **0**.	RowVersioning =1
ShowSystemTables	Specifies whether to treat system catalogs as ordinary SQL tables by default. Value range: **0** and **1** – **0**: The driver does not treat system catalogs as ordinary SQL tables by default. – **1**: The driver treats system catalogs as ordinary SQL tables by default. The default value is **0**.	ShowSystemTables=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
AutoBalance	Specifies whether to enable ODBC load balancing. Value range: **0** and **1** – **0**: Load balancing is disabled. – **1**: Load balancing is enabled. The default value is **0**. **NOTE** If the version of the connected database is earlier than 505.2.0.SPC0200, load balancing is not supported for the standby cluster.	AutoBalance=1
RefreshCNListTime	Time for refreshing the CN list. This parameter can be configured when load balancing is enabled. The value is an integer, in seconds. Value range: 0 to *INT_MAX*. – **0**: This parameter is disabled. – > 0: The set value determines how often the CN list is refreshed. The default value is **10**.	RefreshCNListTime=5
Priority	This parameter can be configured when load balancing is enabled. Value range: **0** and **1** – **0**: This parameter is disabled. – **1**: This parameter is enabled. The default value is **0**. **NOTE** When **Priority** is enabled, all application-initiated connections are preferentially routed to the CNs specified in the configuration file. If none of the configured CNs is available, the connections will be redirected to the remaining CNs.	Priority=1

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
UsingEip	This parameter can be configured when load balancing is enabled. Value range: **0** and **1** – **0**: This parameter is disabled. – **1**: This parameter is enabled. The default value is **0**. **NOTE** This setting determines whether to use service IP addresses (corresponding to the **node_host1** and **node_port1** columns in the pgxc_node system catalog) for load balancing. When **UsingEip** is enabled, service IP addresses are used for load balancing. When it is disabled, data IP addresses (corresponding to the **node_host** and **node_port** columns in the pgxc_node system catalog) are used instead.	UsingEip=1
MaxCacheQueries	Controls the number of prepared statements cached for each connection. Value range: 0 to 4096 The default value is **0**. **NOTE** If this parameter is set to **0**, the prepared statement cache pool is disabled on the client. If this parameter is set to a value greater than 4096, the value **4096** is used. If the number of executed statements exceeds the upper limit specified by **MaxCacheQueries**, the least recently used statements are eliminated.	MaxCacheQueries=128
MaxCacheSizeMiB	Controls the total size of prepared statements cached for each connection. This parameter takes effect when the value of **MaxCacheQueries** is greater than 0. Value range: 0 to 4096 The default value is **1**. **NOTE** If the total size of cached statements is greater than the value of **MaxCacheSizeMiB**, the least recently used statements are eliminated. If this parameter is set to a value greater than 4096, the value **4096** is used. The unit is MB.	MaxCacheSizeMiB=10

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
TcpUserTimeout	Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the **TCP_USER_TIMEOUT** socket option. Value range: 0 to *INT_MAX*. The default value is **0**. **NOTE** The value **0** is used by default. Ignore this parameter for UDS connections. The unit is milliseconds.	TcpUserTimeout=5000
StandbyRead	Specifies whether to enable the read-on-standby mode in a distributed system. Value range: **0** and **1** – **0**: The read-on-standby mode is disabled. – **1**: The read-on-standby mode is enabled. The default value is **0**.	StandbyRead=1
Pqopt	Sets libpq connection parameters. Parameters are separated by spaces. For details about libpq parameters, see **Connection Parameters**.	Pqopt=keepalives=0
KeepaliveTime	The number of seconds of inactivity after which TCP should send a keepalive message to the server. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. If this parameter is set to **0** or is left empty, the default value will be used. **NOTE** To disable the keepalive functionality, set **keepalives** to **0** in Pqopt parameters.	KeepaliveTime=2
KeepaliveInterval	Specifies the interval (in seconds) for retransmitting a TCP keepalive packet until such packet is acknowledged by the target server. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. If this parameter is set to **0** or is left empty, the default value will be used.	KeepaliveInterval=2

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
KeepaliveCount	Controls the number of times that keepalive messages are sent through TCP. If the connection is established using a UDS or if the keepalive functionality is disabled, this parameter will be ignored. Value range: 0 to *INT_MAX*. If this parameter is set to **0** or is left empty, the default value will be used.	KeepaliveCount=2
SocketTimeout	Specifies a socket read/write timeout interval after a connection is successfully established between a client and a server. The unit is seconds. The default value is **0**. **NOTE** This parameter includes the timeout interval for executing database statements. If this parameter is set to a small value, the statement execution may time out. The default value is **0**. Set this parameter based on the actual situation. You are advised to set this parameter to a non-zero value.	SocketTimeout=5
SocketTimeoutInConnect	Controls the socket read/write timeout interval during the connection establishment between a client and a server after a TCP three-way handshake is successful. The unit is seconds. The default value is **5**. **NOTE** Different from **SocketTimeout**, this parameter may block the connection between a client and a server after a three-way handshake is successful. Therefore, you need to set the socket read/write timeout interval to a value that does not affect the statement execution timeout.	SocketTimeoutInConnect=5
CancelTimeout	Specifies the timeout interval for an application to send a cancel message. The unit is seconds. The default value is **0**.	CancelTimeout=5

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
TcpSYNRetries	Controls the maximum number of SYN retransmissions during the TCP three-way handshake. If the connection fails to be established after the number of SYN retransmissions exceeds the threshold, an error is reported. The default value is **0**. **NOTE** This parameter specifies the number of retransmissions due to SYN transmission failures during the three-way handshake when a client establishes a connection on an OS that supports the **TCP_SYNCNT** socket option. The value **0** is used by default. Ignore this parameter for UDS connections.	TcpSYNRetries=3
TextAsLongVarchar	Maps the TEXT type on the kernel to the SQL_LONGVARCHAR or SQL_VARCHAR type on the driver. Value range: **0** and **1**. The value **0** indicates that the TEXT type on the kernel is mapped to the SQL_VARCHAR type on the driver. The value **1** indicates that the TEXT type on the kernel is mapped to the SQL_LONGVARCHAR type on the driver. The default value is **1**.	TextAsLongVarchar=1
MaxLongVarcharSize	Specifies the maximum length of the SQL_LONGVARCHAR type on the driver. Value range: 0 to *INT_MAX*. The default value is **8190**.	MaxLongVarcharSize=8190
MaxVarcharSize	Specifies the maximum length of the SQL_VARCHAR type on the driver. Value range: 0 to *INT_MAX*. The default value is **255**.	MaxVarcharSize=255

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Example
TargetServerType	Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. The setting principles are as follows: – **primary** (default value): Only the primary node in the primary/standby system can be connected. – **cluster-primary**: The connection is limited to CNs in the primary cluster. – **cluster-mainnode**: The connection is limited to CNs in the standby cluster. **NOTE** The **cluster-primary** and **cluster-mainnode** options are supported only in streaming DR scenarios.	TargetServerType=cluster-primary
EnableHostCache	Specifies whether to enable the node cache function. **0** indicates disabled and **1** indicates enabled. Enabled by default.	EnableHostCache=1

Table 7-36 Sslmode options

Sslmode	Enable SSL Encryption	Description
disable	No	SSL connections are not used, which is the default setting.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required, but only data is encrypted. Authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that in the certificate. GaussDB does not support this mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> When establishing connections to the GaussDB server using ODBC, you can enable SSL connections to encrypt client and server communications. To enable SSL connections, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

Step 4 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 5 Run the following command to validate the addition:

```
source ~/.bashrc
```

Step 6 Test the connection.

After the installation, the generated binary file is stored in the **/usr/bin** directory. You can run the **isql -v** *gaussdb* command (*gaussdb* is the data source name).

- If the following information is displayed, the configuration is correct and the connection is successful.

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- If error information is displayed, the configuration is incorrect. Check whether the preceding configuration steps are correctly performed.

> **NOTE**
>
> When ODBC is used to connect to a database, the kernel parameters are set as follows:
>
> ```
> SET extra_float_digits = 2;
> SET DateStyle = 'ISO';
> ```
>
> These parameters may cause the ODBC and gsql clients to display inconsistent data in aspects such as date data display and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.
>
> In M-compatible databases, the default value of **extra_float_digits** is **0**.

----End

Configuring a Data Source in the Windows OS

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

Step 1 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** (64-bit) driver package or **GaussDB-Kernel**_*Database version number*_**Windows_X86_Odbc.tar.gz** (32-bit) driver package, and click **gsqlodbc.exe** to install the driver.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 2 Open the driver manager.

Use the ODBC driver manager corresponding to the ODBC version. If the 64-bit ODBC driver is used, the 64-bit ODBC driver manager must be used. Assume that the OS is installed on drive C (if the OS is installed on another drive, change the path accordingly):

- If you want to use a 32-bit ODBC driver manager in a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not set it by choosing **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

  NOTE

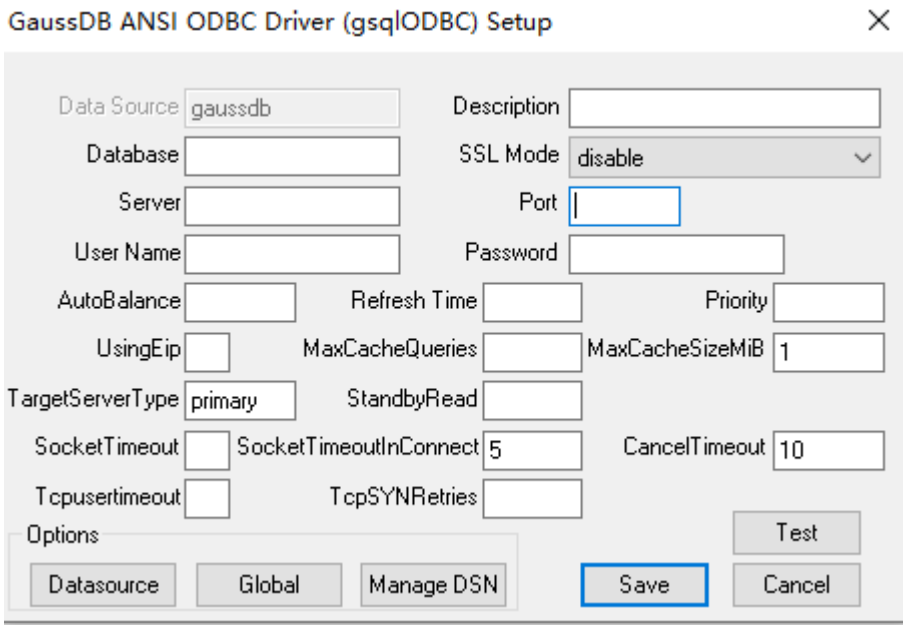
  WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64** stores the 32-bit environment on a 64-bit system. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to open Driver Manager.
- For a 64-bit OS, choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to enable driver management.

Step 3 Configure a data source.

On the **User DSN** tab page of Driver Manager, click **Add** and choose **GaussDB Unicode**. Then set the parameters shown in **Figure 7-44**.

Figure 7-44 ODBC Driver Setup

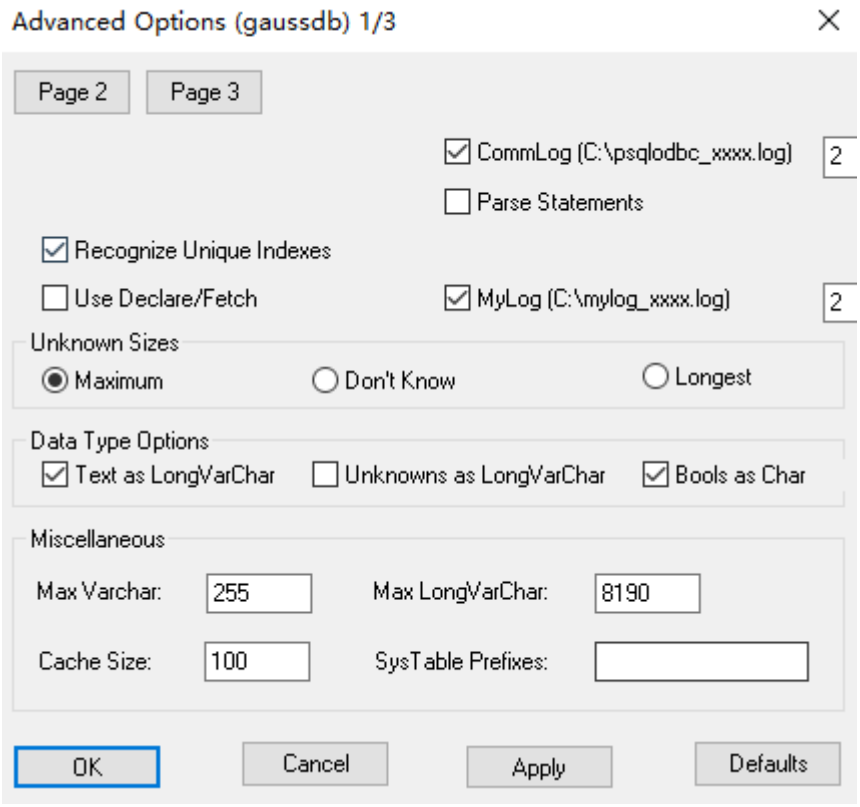


For details about the parameters, see **Configuring a Data Source in the Linux OS**.

You can click **Datasource** to configure whether to print logs. The dialog box shown in **Figure 7-45** is displayed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-45 Advanced Options Page1



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Enable the SSL mode.

Change the value of **SSL Mode** in **Step 3** to **require**.

Table 7-37 Sslmode options

Sslmode	Enable SSL Encryption	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Sslmode	Enable SSL Encryption	Description
require	Yes	SSL connection is required, but only data is encrypted. Authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support the cert authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that in the certificate. Currently, Windows ODBC does not support the cert authentication.

NOTE

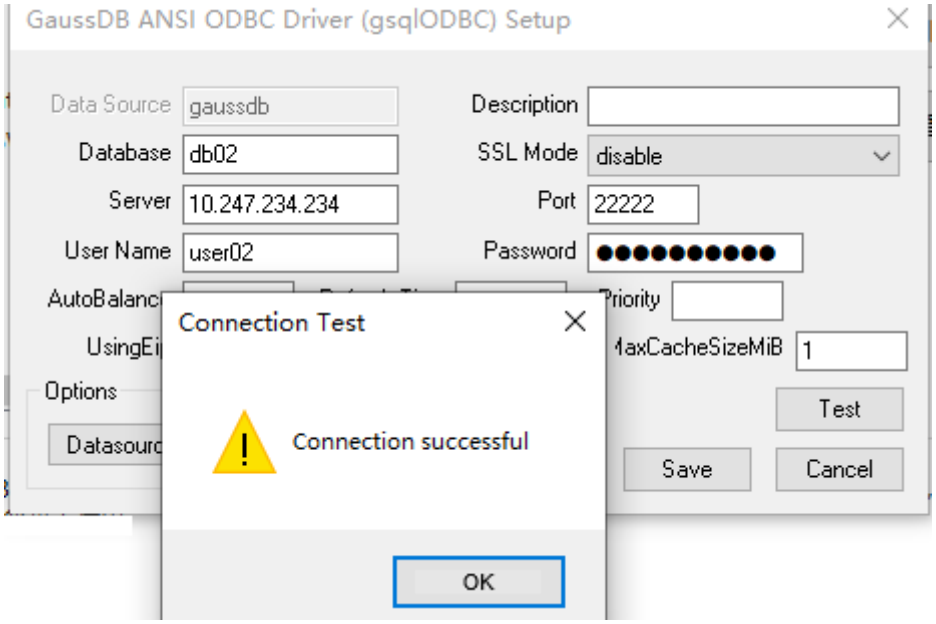
When establishing connections to the GaussDB server using ODBC, you can enable SSL connections to encrypt client and server communications. To enable SSL connections, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

Step 5 Test the connection.

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection is successful.

Figure 7-46 Successful configuration



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If error information is displayed, the configuration is incorrect. Check whether the preceding configurations are correct.

NOTE

When ODBC is used to connect to a database, the kernel parameters are set as follows:

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

These parameters may cause the ODBC and gsql clients to display inconsistent data in aspects such as date data display and floating-point precision representation. If the result is not as expected, you are advised to explicitly set these parameters in the ODBC application code.

In M-compatible databases, the default value of **extra_float_digits** is **0**.

----End

Example

The following uses the source program **DBtest.c** as an example to describe how to obtain and process data in GaussDB through ODBC.

Prerequisites: The data source has been configured successfully. (For details about the complete example, see **Obtaining and Processing Data in a Database**.)

```
// DBtest.c (compile with: libodbc.so)
// For details about program header files and global variables, see the complete example.

// Allocate an environment handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   printf("Error AllocHandle\n");
   exit(0);
}

// Set the version information (environment attributes).
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

// Allocate a connection handle.
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
   exit(0);
}

// Obtain the username and password.
char *userName;
userName = getenv("EXAMPLE_USERNAME_ENV");
char *password;
password = getenv("EXAMPLE_PASSWORD_ENV");

// Set connection properties.
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

// Connect to the database. userName and password indicate the username and password for connecting to the database, respectively.
// If the username and password have been set in the odbc.ini file, you can retain "". However, you are advised not to do so because the username and password will be disclosed if the permission on 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);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
  exit(0);
}
printf("Connected !\n");
```

7.25.2.3.4 Executing SQL Statements

NOTE

- ODBC connects an application to a database and delivers the SQL statements sent by the application to the database. It does not parse the SQL syntax. Therefore, when confidential information (such as a plaintext password) is written into the SQL statement sent by an application, the confidential information is exposed in the driver log.
- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

The following provides a brief example of executing SQL statements (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// Set statement attributes.
SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3, 0);

// Allocate a statement handle.
SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

// Run an SQL statement directly.
SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25, 'li')", SQL_NTS);

// Prepare for execution.
SQLPrepare(V_OD_hstmt, "insert into customer_t1 values(?)", SQL_NTS);

// Bind parameters.
SQLBindParameter(V_OD_hstmt,1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0,
     &value, 0, NULL);

// Run the prepared statement.
SQLExecute(V_OD_hstmt);      SQLExecDirect(V_OD_hstmt, "select c_customer_sk from customer_t1",
SQL_NTS);
```

7.25.2.3.5 Processing the Result Set

ODBC retrieves data from the database and provides the data to an application for processing. The processing includes but is not limited to: retrieving data, displaying data, processing data, transmitting data, and implementing service logic.

The following provides a brief example (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// After an SQL statement is executed, retrieve the attributes of a column in the result set.
SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
printf("SQLColAttribute %s\n",typename);

// Bind the result set.
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
  (SQLLEN *)&V_OD_err);

// Use SQLFetch to retrieve data from the result set.
V_OD_erg=SQLFetch(V_OD_hstmt);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
// Use SQLGetData to retrieve and return data.
while(V_OD_erg != SQL_NO_DATA)
{
  SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
  printf("SQLGetData ----ID = %d\n",V_OD_id);
  V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");
```

7.25.2.3.6 Closing a Connection

The following provides a brief example of closing a database connection (refer to **Obtaining and Processing Data in a Database** for the complete example):

```
// 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);
```

7.25.2.3.7 Best Practices

GaussDB provides best practices for ODBC.

For details, see "Best Practices for ODBC" in *Best Practices*.

7.25.2.4 Typical Issues

Error Message "[UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/gsqlodbcw.so':file not found" Is Displayed When a Data Source Is Connected

The possible causes and solutions are as follows:

- The path configured in the **odbcinst.ini** file is incorrect.

  Run **ls** to check the path in the error information and verify that the **gsqlodbcw.so** file exists and you have the execute permission on it.

- The dependency library of **gsqlodbcw.so** does not exist or is not in system environment variables.

  Run the **ldd** command to check the path in the error information. If the unixODBC library such as **libodbc.so.1** is missing, reconfigure unixODBC by referring to **Creating a Database Connection**, and ensure that the **lib** directory in the installation path is added to *LD_LIBRARY_PATH*. If the problem persists after reinstallation, manually copy the content in the **unixodbc/lib** directory of the database installation package to the **lib** directory in the installation path of the unixODBC. If other libraries do not exist, add the **lib** directory under the ODBC driver package to *LD_LIBRARY_PATH*. If other standard libraries are missing, install them.

Error Message "[UnixODBC]connect to server failed: no such file or directory." Is Displayed When a Data Source Is Connected

The possible causes and solutions are as follows:

- An incorrect or unreachable database IP address or port was configured.

  Check the **Servername** and **Port** configuration items in the data source.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Server monitoring is improper.

  If **Servername** and **Port** are correctly configured, ensure that the proper NIC and port are monitored by following the database server configurations in **Configuring a Data Source in the Linux OS** or **Configuring a Data Source in the Windows OS**.

- Firewall and network gatekeeper settings are improper.

  Check firewall settings and ensure that the database communication port is trusted.

  Check to ensure that network gatekeeper settings are proper (if any).

Error Message "authentication method 10 not supported." Is Displayed

The possible causes are as follows:

This problem may occur when an open-source client is used. The database stores only the SHA-256 hash value of the password, but the open-source client supports only MD5 hash values.

NOTE

- The database stores the hash values of user passwords instead of actual passwords.
- If a password is updated or a user is created, both types of hash values will be stored, compatible with open-source authentication protocols.
- An MD5 hash value can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash value. Passwords in the source version will only have SHA-256 hash values and not support MD5 authentication.
- The MD5 cryptographic algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure cryptographic algorithm.

To solve this problem, you can update the user password or create a user with the same permissions as the original user and use the new user to connect to the database.

Error "Driver's SQLAllocHandle on SQL_HANDLE_DBC failed." Is Displayed

The possible causes are as follows:

The executable file (such as the **isql** of unixODBC) and the database driver (**gsqlodbcw.so**) depend on different library versions of ODBC, such as **libodbc.so.1** and **libodbc.so.2**. You can verify this problem as follows:

```
ldd `which isql` | grep odbc
ldd gsqlodbcw.so | grep odbc
```

If the suffixes of the output **libodbc.so** files are different or point to different physical files on the disk, a version conflict occurs. Both **isql** and **gsqlodbcw.so** require **libodbc.so** to be loaded. If they load different physical files, two sets of function lists with the same name are generated in a visible domain (the **libodbc.so.***** function export lists of unixODBC are the same). This results in conflicts and the database driver cannot be loaded.

Solution:

Retain the required unixODBC version, uninstall the conflicting version (for example, **libodbc.so.2**), and create a soft link with the same name but the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

suffix **.so.2** for the retained **libodbc.so.1** file (that is, the new **libodbc.so.2** file points to **libodbc.so.1**).

Garbled Characters in Encoding

The possible causes are as follows:

The character sets of the client and server are inconsistent. For example, the client uses GBK and the server uses UTF-8.

Solution:

The character sets of the application, ODBC driver, and database must be the same.

Error "unsupported frontend protocol 3.51: server supports 1.0 to 3.0." Is Displayed

The possible causes are as follows:

The database version is too early or the database is an open-source database.

Solution:

Use the driver of the required version to connect to the database.

7.26 Online Scale-out Based on Hash Bucket Tables

7.26.1 Overview

Description

As the cluster deployment time and services increase, the database gradually reaches its bottleneck in terms of running performance and storage and requires more hosts to improve the cluster performance and storage capability to ensure stable service running. GaussDB provides gs_expand to add hosts and redistribute data.

Scenarios

As services grow, the cluster performance and storage capacity reach the bottleneck. In this case, more hosts are required to improve the cluster performance and storage capacity to ensure continuous and stable service running. Offline scale-out blocks services. Therefore, online scale-out is required to ensure that services are running properly during scale-out and redistribution.

Specifications and Limitations

- During cluster scale-out, do not change the value of **expandimplolap_step/expandimplolap_step.dat** under *$PGHOST*. Otherwise, re-executing scale-out and redistribution will be affected.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Parameter configuration and physical resource allocation for new DNs must keep consistent with those of the current DNs. If more resources are available on the new servers, you can configure more DNs.
- The new hosts must meet the cluster installation requirements. The overall performance of the new cluster depends on the host with the lowest configuration in the cluster. The limitations are as follows:
  - For a two-copy cluster, the number of hosts to be added must be greater than or equal to 2.
  - For other types of clusters, the number of hosts to be added must be greater than or equal to 3.
  - The locale and encoding of the new hosts must be the same as those of the existing hosts.
  - The cluster installation path and user group of the new hosts have been planned.
- The OSs of new hosts in the cluster must be of the same major version as that of the existing hosts. OSs can be different in minor versions.
- If the hardware specifications (memory and hard disk) of new hosts are lower than those of the existing hosts, you need to gradually replace the low-performance servers with better servers in the future.
- A new scale-out can be performed only after the previous one is complete. For example, no more scale-out is allowed if the previous scale-out is interrupted by a power outage. In this case, you need to recover and complete the previous scale-out first.
- If a third-party plug-in exists, you need to manually copy related dependencies to the new node after scale-out.
- During heterogeneous scale-out (the hardware specifications of the new servers are different from those of the existing servers in the cluster), the software and hardware specifications of servers in the same batch must be the same.
- If the database version is inconsistent with the OM version (the database code and OM code come from different software packages), scale-out is not supported.
- Resource usage during scale-out:
  - In the ultimate performance scenario, that is, the number of buckets in each batch is 1024 and a maximum of 1024 buckets can be brought online at a time, the I/O resource usage is not controlled.
  - In the online scenario (default parameter configuration), that is, the number of buckets in each batch is 64 and the maximum number of buckets in each batch is 64, the maximum disk I/O usage is 20%.
  - I/O and incremental Xlogs occupy a small amount of disk space and are affected by the number of retained Xlogs.
  - Online index creation limitations apply to CBIs if any during online index type conversion, which may affect service performance and cause resource contention.

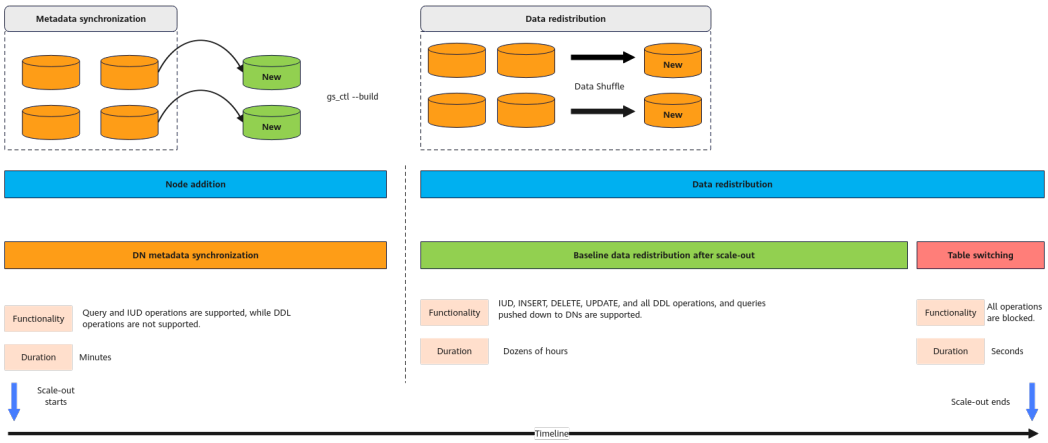
Technical Architecture

The online scale-out process consists of two core phases: cluster node addition and data redistribution.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Cluster node addition: Basic metadata is synchronized to the new DNs or CNs.
- Data redistribution: Data is migrated from the old cluster to the new cluster.

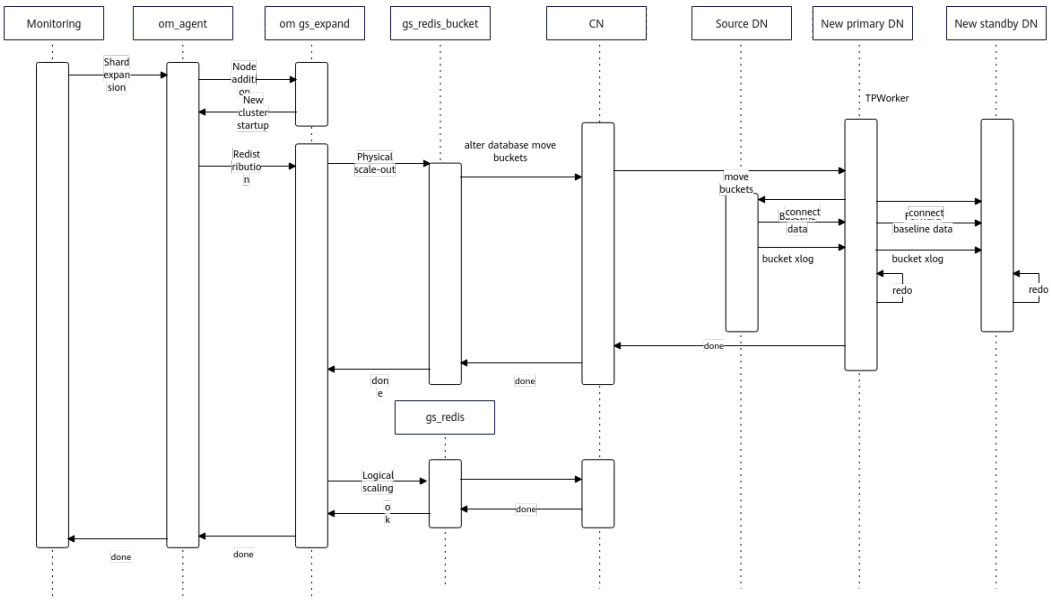
Figure 7-47 Technical architecture



7.26.2 Technical Principles

Core Mechanism

Figure 7-48 Core Mechanism



NOTE

gs_expand integrates physical and logical scale-out. Therefore, the figure contains information about logical scale-out. You are advised not to use bucket tables and non-bucket tables together in a database during physical scale-out.

Performance Metrics

For DNs in a cluster, assume that the read bandwidth is 500 MB/s, the write bandwidth is 800 MB/s, and a 10GE network is configured. When each DN

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

exclusively occupies a disk, the service load is low, and resource control is not performed (the resource control level is **f**, the concurrency of operations on tables is **8**, and the scan concurrency is **4**), the time required for doubling the capacity of 1 TB data on a single-DN shard is as follows:

- Ordinary tables: 3 hours.
- Hash bucket tables: 2 hours (The number of buckets in each batch is 1024, and a maximum of 1024 buckets can be brought online at a time.)

API Description

- gs_expand

  gs_expand adds hosts to improve the overall processing capability and storage capacity of the cluster when the cluster performance and storage resources reach the bottleneck during service scale-out and data volume increase.

  NOTE

  - You are advised to use this API to add nodes and redistribute data.

- gs_redis_bucket

  gs_redis_bucket, mainly called by the OM script, redistributes data in hash bucket tables in the cluster after scale-out.

  NOTE

  - This API is not recommended. You are advised to use the gs_expand tool to redistribute data.

- gs_redis

  gs_redis tool, mainly called by the OM script, redistributes data in non-hash bucket tables in the cluster after scale-out.

  NOTE

  - This API is not recommended. You are advised to use the gs_expand tool to redistribute data.

- System functions

Function	Description
gs_redis_get_plan	Obtains the complete migration plan. It can be called only by administrators on CNs during scale-out.
gs_redis_get_bucket_statistics	Obtains the log stream transmission status. It can be called only by administrators during scale-out.
gs_redis_set_distributed_db	Sets the database that is being redistributed in hash bucket mode. It can be called only by administrators on DNs during scale-out.
gs_redis_hashbucket_update_segment_header	Updates the headers of all hash bucket tables in the current database. It can be called only by administrators on CNs during scale-out.
gs_redis_local_get_segment_header	Obtains the header of a segment-page table. It can be called only on DNs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Description
gs_redis_local_update_segment_header	Updates the header of a segment-page table. It can be called only on DNs.
gs_redis_hashbucket_update_inverse_pointer	Updates the inverse pointers of the current batch of buckets in all hash bucket tables in the current database. It can be called only by administrators on CNs during scale-out.
gs_redis_local_get_inverse_pointer	Obtains inverse pointers. It can be called only by administrators on DNs during scale-out.
gs_redis_local_update_inverse_pointer	Records Xlogs for updating inverse pointers. It can be called only by administrators on DNs during scale-out.
gs_redis_local_set_hashbucket_frozenxid	Changes the value of **relfrozenxid64** of a hash bucket table in the system catalog. It can be called only by administrators on DNs during scale-out.
gs_redis_set_hashbucket_frozenxid	Changes the value of **relfrozenxid64** of a hash bucket table on a new DN in the system catalog. It can be called only by administrators on CNs during scale-out.
gs_redis_set_nextxid	Changes the value of **next_xid** on a DN. It can be called only during scale-out. It can be called only by administrators on DNs.
gs_redis_set_csn	Changes the value of **next_csn** on a DN. It can be called only by administrators on DNs during scale-out in GTM-free mode.
gs_redis_check_bucket_flush	Queries whether the private buffer has been fully flushed during RTO replay. It can be called only by administrators during scale-out.
gs_redis_get_flush_page_lsn	Queries the page flushing information replayed during bucket scale-out. It is not available in releases and can be called only by administrators during scale-out.
gs_redis_show_bucketxid	Queries the **bucketxid** corresponding to a specified bucket. It can be called only by administrators on DNs during scale-out.
gs_redis_drop_bucket_files	Drops the files that have been physically migrated from the source node. It can be called only by administrators on CNs during scale-out.
gs_redis_local_drop_bucket_files	Drops the corresponding bucket list. It can be called only by administrators on DNs during scale-out.
gs_redis_local_drop_bucket_files	Drops the bucket list of the corresponding database from the range bucket table. It can be called only by administrators on DNs during scale-out.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Description
gs_redis_get_undo_state	Obtains the undo status during scale-out and redistribution. It can be called only by administrators during scale-out.

NOTE

- For details, see "SQL Reference > Functions and Operators > System Administration Functions > Bucket System Functions" in *Reference*.
- The preceding system functions are not recommended. You are advised to use the gs_expand tool to add nodes and redistribute data.

- Tables

  The following tables are created during logical scale-out.

Table	Description
redis_progress_detail	Records the execution time of scale-out and redistribution of non-hash bucket tables.
redis_status	Records the overall redistribution progress of the cluster.
pgxc_redistb	Records information about user tables to be redistributed. The information is automatically deleted after scale-out and redistribution are complete.

7.26.3 Operation Guide

Prerequisites

- The cluster status must be **Normal** or **Degraded**. The **Degraded** status is acceptable only due to an instance fault. For the reason of VM exceptions (non-software exceptions), such as disk faults and communication faults, this status is not acceptable.
- The network connection (even for the abnormal instance) is normal.
- Instance status: All CNs are normal. At least the primary GTM and CM Server nodes are alive. The majority of ETCD instances are normal.
- The DN instance status can be any of the following:
  - In a two-copy cluster, at least the primary DN in each shard is normal. (For hash bucket table scale-out, both primary and standby DNs must be normal.)
  - In a two-AZ cluster, at least the primary DN and one standby DN in each shard are alive.
  - In other cases, the number of abnormal DNs in each shard must be less than half.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The cluster is not locked, and information in the configuration file is correct and consistent with the current cluster configuration.
- The preinstallation script gs_preinstall has been executed based on the cluster configuration file for scale-out. That is, the preinstallation must have been completed.
- The communication between the new hosts and the existing cluster has been established and the network is normal.
- All the CNs can be connected. You can run the **gs_check -i CheckDBConnection** command as a cluster user to check whether a CN is connected.

```
Parsing the check items config file successfully
Distribute the context file to remote hosts successfully
Start to health check for the cluster. Total Items:1 Nodes:3

Checking...              [=========================] 1/1
Start to analysis the check result
Start to analysis the item CheckDBConnection
analysis item: CheckDBConnection, use time: 0.021239858004264534
Analysis the item CheckDBConnection finished
AnalysisResult total time: 0.021775729983346537
CheckDBConnection.........................OK
The item run on 3 nodes.  success: 3

Analysis the check result successfully
Success.    All check items run completed. Total:1    Success:1
For more information please refer to /script/gspylib/inspection/output/
CheckReport_20250508348994120 2.tar.gz
```

- Before scale-out, ensure that each database has a public schema. Otherwise, the scale-out will fail.
- Before scale-out, the logical replication slots on each CN and DN must have been deleted.

```
Run the following command on each CN and DN to obtain all logical replication slots:
gaussdb=# SELECT * FROM pg_catalog.pg_get_replication_slots() WHERE slot_type='logical';

Run the following command on each CN and DN to delete the queried logical replication slots:
gaussdb=# SELECT * FROM pg_drop_replication_slot('slot_name');
```

- After the scale-out, none of the new nodes contains the ETCD component.
- Client connections for creating temporary tables must be disconnected before scale-out, because the temporary tables will be invalid during and after the scale-out. Operations on the tables will become invalid as well.
- If the number of objects (data tables, views, and indexes) in the database exceeds 500,000, you are advised to set max_prepared_transactions and max_locks_per_transaction to 1000 and 512 for CNs and DNs respectively before scale-out to improve performance and avoid memory problems. If the values of max_prepared_transactions and max_locks_per_transaction are greater than the recommended values, you do not need to set them. For details about the GUC parameter **max_prepared_transactions**, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. For details about the **max_locks_per_transaction** parameter, see "Database GUC Parameters > GUC Parameters > Lock Management" in *Reference*.
- Before scale-out, check the value of **comm_max_datanode** of the existing cluster.
  - Use gsql to connect to the CN and run **show comm_max_datanode**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - If the value of **comm_max_datanode** is less than the number of DNs in the cluster after scale-out, you need to set the **comm_max_datanode** parameter. For details, see "Database GUC Parameters > GUC Parameters > Connection and Authentication > Communications Library Parameters" in the *Reference*. If the original cluster has 128 DNs and 128 DNs are added, set the comm_max_datanode parameter to 256 before scale-out.
- After the new cluster is started, the system cannot be rolled back to the status before scale-out.
- During redistribution, objects used for resource control, that is, **redisuser**, **redisrespool**, **redisclass**, and **redisgrp**, cannot be occupied. If the **redisuser** user exists in the cluster, an error will be reported and the scale-out exits. In addition, the **redisuser** user cannot be created during the scale-out.
- The following operations are prohibited during the redistribution:
  - Binding other users to the resource pool **redisrespool** dedicated for redistribution;
  - Running **gs_cgroup -M** to damage the current Cgroup configuration. If you have run the **gs_cgroup --revert** command to restore the default settings of a faulty Cgroup, restart the cluster to make the new **redisclass** and **redisgrp** take effect.
- If OLTP table compression is enabled for hash bucket tables in the current cluster, wait until all compression tasks are complete and disable automatic scheduling. Compression scheduling is not supported during scale-out.
- Before redistribution, check whether there are residual invalid indexes. If the return value is greater than or equal to 1, run the following **ALTER INDEX** command to invalidate indexes:

```
gaussdb=# select count(*) from pg_index where indisvalid = 'f' and indisusable = 't';
 count
-------
     0
(1 row)
gaussdb=# ALTER INDEX i1 UNUSABLE;
ALTER INDEX
```

Syntax

- Scale out a cluster.

```
gs_expand -t dilatation -X XMLFILE [-l LOGFILE] [--parallel-jobs=NUMBER] [--time-out=SECS] [--dilatation-mode=MODE] {--nodegroup-name=LCGROUPNAME | --addto-elastic-group}
```

- Set redistribution parameters.

```
gs_expand -t parametercfg [--lockwait-timeout=MSEC] [--lockwait-interval=SECS] [--trylock-threshold=NUMBER] [--enable-cancel=Boolean] [--last-catchup-threshold=MSEC] [--catchup-times=NUMBER] [--write-error-mode=Boolean] [--catchup-query-dop=NUMBER] [--parallel-catchup-threshold=NUMBER] [--parallel-reindex-jobs=NUMBER][--redis-retry-times=NUMBER][--redis-retry-interval=NUMBER][--redis-bucket-parse-redo-num=NUMBER][--redis-bucket-workers-per-paser=NUMBER][--redis-bucket-keep-bucketxlogs=NUMBER]
```

- Data redistribution

```
gs_expand -t redistribute  [--parallel-jobs=NUMBER] [--resource-level=f/h/m/l] [--failure-limit=NUMBER] [--build-redistb] [--redis-mode=MODE] [-l LOGFILE] [--priority-tables=FILE] [--exclude-tables=FILE] [--time-out=SECS] [--max-worker] [--max-dop] [--join-tables=FILE]
```

- Perform rollback.

```
gs_expand -t crashclean -X XMLFILE [-l LOGFILE]
```

Precautions for Physical Scale-Out

- Node groups cannot be bound when the cluster is redistributing data (that is, after nodes are added and new node groups are created).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Some indexes and GSIs in the expression index form on hash bucket tables are rebuilt offline during redistribution. During the rebuilding, services are blocked. GSIs that are not partial indexes or expression indexes are rebuilt online during redistribution, which does not block services. GSIs are valid during scale-out and redistribution.
- During scale-out of hash bucket tables, metadata must be updated by traversing databases before node groups are created. If DROP DATABASE operations are frequently performed when new nodes are added, an error will be reported when the number of retries exceeds the upper limit. In this case, you need to perform scale-out again after such DDL operations on the cluster are complete.
- If residual partitions with **wait_clean_gpi** set to **y** are detected during hash bucket table scale-out, the error message "Please manually perform the 'vacuum' operation on these databases." is displayed. You need to connect to the database as prompted, run the **VACUUM** command, and perform scale-out again.
- If the primary DN (in the old or new DN shards) or a standby DN in the new shards is faulty during redistribution, the redistribution fails and exits. In this case, you need to perform redistribution again.
- In the current version, hash bucket table scale-out can be performed when DNs in the new shards are faulty before redistribution but the number of available primary and standby DNs exceeds half of the total number of copies in the respective shard.
  - In scenarios where one primary DN and multiple standby DNs are deployed, scale-out is supported as long as the majority requirement is met. For example, if one primary DN and two standby DNs are deployed, scale-out is supported when one standby DN is faulty.
  - In the scenario where one primary DN, one standby DN, and one logger are deployed, scale-out is supported when only the logger is faulty. If the standby DN is faulty, scale-out is not supported.
  - In the scenario where one primary node and one standby node are deployed, scale-out is supported only when both the primary and standby nodes are available.
- In the current version, CREATE TABLE, DROP TABLE, and TRUNCATE TABLE can be performed on hash bucket tables during scale-out. Other DDL operations are not supported.
  - If CREATE, DROP, or TRUNCATE performed on the target hash bucket table involves cascading index operations (except the TOAST index), the table-level DDL operation fails.
  - A hash bucket partitioned table can be created, dropped, or truncated, but such operations cannot be performed on a partition of the table.
  - When metadata is updated during scale-out or the inverse pointer is updated during bucket redistribution, if a table is dropped, the preceding operations fail due to the window period. In this case, you need to perform scale-out again.
- You are advised not to delete CBIs or tables with CBIs during scale-out. Otherwise, redistribution may fail and you need to perform the scale-out again.
- During hash bucket table scale-out, you cannot connect to CNs to call global_space_shrink to scale in segment-page files. Considering resource

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

contention, you are advised not to directly connect to DNs to scale in segment-page files.

- During hash bucket table scale-out, parameters in read-only mode are ignored and the online execution mode is still used. During scale-out, the **redisuser** user is used to perform redistribution to prevent scale-out failures caused by residual threads. In addition, automatic retry is not supported.
- The following features are not supported during hash bucket table scale-out: logical replication, fine-grained backup and restoration, online DN scale-in, subpartitioning, and node group-level migration.
- During hash bucket table scale-out, transactions involving CBIs of Ustore hash bucket tables are cached in the memory before being committed and the cache is released after the transactions are committed. If the accumulated memory usage exceeds 1 GB, the scale-out process will be terminated, reporting an error. After an error will be reported, you can re-execute the scale-out process to retry.
- If a flashback query attempts to access the snapshot generated during or before the hash bucket table scale-out, the error "snapshot csn xxx less or equal to bucketcsn xxx from old DN" may be reported because the CSN is too old.
- If a query attempts to access the data on a new node that goes online after migration based on a long transaction snapshot generated before the scale-out, the error "snapshot csn less or equal to bucketcsn from old DN" may be reported.

Procedure

NOTE

- Ensure that a distributed cluster is available.
- The following describes how to modify the cluster deployment configuration file (for example, **clusterconfig.xml**) as user **omm**.
- The online scale-out process consists of the cluster node addition phase and redistribution phase.
  - **Step 1** to **Step 7** are the cluster node addition phase.
  - **Step 8** is preparing for redistribution.
  - **Step 9** to **Step 11** are the redistribution phase.

Step 1 Modify the cluster deployment configuration file **clusterconfig.xml** and add configuration information about the hosts to be added.

1. Add the names of all new hosts to the end of the **nodeNames** field in the cluster information (the sequence of existing host names remains unchanged), and add the names of all new hosts to the **sqlExpandNames** field. For example, **gaussdb04**, **gaussdb05**, and **gaussdb06** are the names of the new hosts, and **192.168.255.253**, **192.168.255.254**, and **192.168.255.255** are the corresponding IP addresses.

```
<CLUSTER>
<PARAM name="clusterName" value="mppdbCluster" />
<PARAM name="nodeNames"
value="gaussdb01,gaussdb02,gaussdb03,gaussdb04,gaussdb05,gaussdb06" />
<PARAM name="backIp1s"
value="192.168.255.250,192.168.255.251,192.168.255.252,192.168.255.253,192.168.255.254,192.168.255.255" />
<PARAM name="gaussdbAppPath" value="/data/lixy1/install/app" />
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
<PARAM name="gaussdbLogPath" value="/data/lixy1/install/log" />
<PARAM name="gaussdbToolPath" value="/data/lixy1/tool/wisequery" />
<PARAM name="clusterType" value="single-primary-multi-standby"/>
<PARAM name="sqlExpandNames" value="gaussdb04, gaussdb05, gaussdb06"/>
</CLUSTER>
```

2. Add the instance information about the new hosts to the **clusterconfig.xml** file.

```
<!-- Information about the node deployment on plat4 -->
      <DEVICE sn="1000004">
         <PARAM name="name" value="gaussdb04"/>
         <PARAM name="azName" value="AZ1"/>
         <PARAM name="azPriority" value="1"/>
         <!-- If only one NIC is available for the server, set backIP1 and sshIP1 to the same IP address.
-->
         <PARAM name="backIp1" value="192.168.255.253"/>
         <PARAM name="sshIp1" value="192.168.255.253"/>
         <!-- cm -->
         <PARAM name="cmServerPortStandby" value="25500"/>
         <PARAM name="cmDir" value="/data/lixy1/install/cmserver"/>

         <!-- cn -->
         <PARAM name="cooNum" value="0"/>
         <PARAM name="cooPortBase" value="25108"/>
         <PARAM name="cooListenIp1" value="192.168.255.253"/>
         <PARAM name="cooDir1" value="/data/lixy1/install/coordinator"/>
         <!--etcd-->
         <!--DN-->
         <PARAM name="dataNum" value="1"/>
         <PARAM name="dataPortBase" value="26600"/>
         <PARAM name="dataNode1" value="/data/lixy1/install/master4,gaussdb05,/data/lbs2/install/
slave4_1,gaussdb06,/data/lbs3/install/slave4_2"/>
      </DEVICE>
<!-- Information about the node deployment on plat5 -->
      <DEVICE sn="1000005">
         <PARAM name="name" value="gaussdb05"/>
         <PARAM name="azName" value="AZ1"/>
         <PARAM name="azPriority" value="1"/>
         <!-- If only one NIC is available for the server, set backIP1 and sshIP1 to the same IP address.
-->
         <PARAM name="backIp1" value="192.168.255.254"/>
         <PARAM name="sshIp1" value="192.168.255.254"/>
         <!-- cm -->
         <PARAM name="cmServerPortStandby" value="25500"/>
         <PARAM name="cmDir" value="/data/lixy1/install/cmserver"/>

         <!-- cn -->
         <PARAM name="cooNum" value="0"/>
         <PARAM name="cooPortBase" value="25108"/>
         <PARAM name="cooListenIp1" value="192.168.255.254"/>
         <PARAM name="cooDir1" value="/data/lixy1/install/coordinator"/>
         <!--etcd-->
         <!--DN-->
         <PARAM name="dataNum" value="1"/>
         <PARAM name="dataPortBase" value="26800"/>
         <PARAM name="dataNode1" value="/data/lixy1/install/master5,gaussdb04,/data/lbs2/install/
slave5_1,gaussdb06,/data/lbs3/install/slave5_2"/>
      </DEVICE>
<!-- Information about node deployment on plat6 -->
      <DEVICE sn="1000006">
         <PARAM name="name" value="gaussdb06"/>
         <PARAM name="azName" value="AZ1"/>
         <PARAM name="azPriority" value="1"/>
         <!-- If only one NIC is available for the server, set backIP1 and sshIP1 to the same IP address.
-->
         <PARAM name="backIp1" value="192.168.255.255"/>
         <PARAM name="sshIp1" value="192.168.255.255"/>
         <!-- cm -->
         <PARAM name="cmServerPortStandby" value="25500"/>
         <PARAM name="cmDir" value="/data/lixy1/install/cmserver"/>
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        <!-- cn -->
        <PARAM name="cooNum" value="0"/>
        <PARAM name="cooPortBase" value="25108"/>
        <PARAM name="cooListenIp1" value="192.168.255.255"/>
        <PARAM name="cooDir1" value="/data/lixy1/install/coordinator"/>
        <!--etcd-->
        <!--DN-->
        <PARAM name="dataNum" value="1"/>
        <PARAM name="dataPortBase" value="27000"/>
        <PARAM name="dataNode1" value="/data/lixy1/install/master6,gaussdb04,/data/lbs2/install/
slave6_1,gaussdb05,/data/lbs3/install/slave6_2"/>
    </DEVICE>
```

NOTE

- New nodes cannot contain primary/standby GTMs and CM Servers.
- DNs on a new node must be deployed in a ring to prevent the original primary/standby relationship from being broken.

Step 2 Log in to a server as user **root**.

Step 3 Go to the directory where the installation package is stored. For example, the installation package is stored in **/opt/software/gaussdb**. Replace it with the actual path based on the environment.

```
cd /opt/software/gaussdb
```

Step 4 Prepare the environment, decompress the installation package, go to the **script** directory in the directory where the installation package is stored, and perform preinstallation as the **root** user.

```
./gs_preinstall -U omm -G dbgrp -X /opt/software/gaussdb/clusterconfig_dilatation.xml --alarm-type=5
Parsing the configuration file.
Successfully parsed the configuration file.
Check the preinstallation result file.
Successfully delete task result file
Installing the tools on the local node.
Successfully installed the tools on the local node.
Are you sure you want to create trust for root (yes/no)?yes
Please enter password for root
Password:
Successfully created SSH trust for the root permission user.
Setting HOST_IP env
Successfully set HOST_IP env.
set hotpatch env.
clear hotpatch env.
Distributing package.
Successfully distribute local lib to remote pylib.
Begin to distribute package to tool path.
Successfully distribute package to tool path.
Begin to distribute package to package path.
Successfully distribute package to package path.
Successfully distributed package.
Are you sure you want to create the user[omm] and create trust for it (yes/no)? yes
Preparing SSH service.
Successfully prepared SSH service.
Installing the tools in the cluster.
Successfully installed the tools in the cluster.
Checking hostname mapping.
Successfully checked hostname mapping.
Creating SSH trust for [omm] user.
Please enter password for current user[omm]
Password:
Successfully created SSH trust for [omm] user.
Checking OS version.
Successfully checked OS version.
Creating cluster's path.
Successfully created cluster's path.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Set and check OS parameter.
Setting OS parameters.
Successfully set OS parameters.
Warning: Installation environment contains some warning messages.
Please get more details by "/opt/software/gaussdb/script/gs_checkos -i A -h
gaussdb01,gaussdb02,gaussdb03,gaussdb04,gaussdb05,gaussdb06".
Set and check OS parameter completed.
Preparing CRON service.
Successfully prepared CRON service.
Setting user environmental variables.
Successfully set user environmental variables.
Configuring alarms on the cluster nodes.
Successfully configured alarms on the cluster nodes.
Setting the dynamic link library.
Successfully set the dynamic link library.
Setting pssh path
Successfully set pssh path.
Setting Cgroup.
Successfully set Cgroup.
Set ARM Optimization.
Successfully set ARM Optimization.
Setting finish flag.
Scene: non-huaweiyun
Successfully set finish flag.
Preinstallation succeeded.
Check the preinstallation result file.
Modify the permission on the preinstallation result file.
```

Step 5 Switch to the cluster user **omm**.

```
su - omm
```

Step 6 Set the node addition and redistribution parameters.

If the scale-out and redistribution are performed during off-peak or idle hours, you can set **redis-bucket-reindex-cbi-offline** to **true** and run the following command; otherwise, you are advised to skip this setting:

```
# Set the CBI rebuild mode. If there are off-peak service hours, enable this parameter to enhance CBI
rebuild performance in offline scale-out scenarios.
gs_expand -t parametercfg --redis-bucket-reindex-cbi-offline=true
Redis parameters are set successfully.
```

Step 7 Add nodes to the cluster.

```
# You are advised to use the default parameter values.
gs_expand -t dilatation -X /opt/software/gaussdb/clusterconfig_dilatation.xml
Check cluster version consistency.
Successfully checked cluster version.
Distributing cluster dilatation XML file.
Checking static configuration.
Static configuration is matched with the old configuration file.
Checking the cluster status.
Performing health check.
Health check succeeded.
Backing up parameter files.
Successfully backed up parameter files.
Installing.
Deleting instances from ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Uninstalling applications on ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Successfully uninstalled applications on ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Checking installation environment on all nodes.
Installing applications on all nodes.
Resume om hotpatch.
Successfully resume om hotpatch.
Synchronizing alarmItem configuration to new nodes.
Successfully synchronized alarmItem configuration to new nodes.
Installation is completed.
Configuring.
Configuring new nodes.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Checking node configuration on all nodes.
Initializing instances on all nodes.
Configuring pg_hba on all nodes.
Successfully configured new nodes.
Begin scpCertFile.
ScpCertFile is completed.
Rebuilding new nodes.
Locking cluster.
Successfully locked cluster.
Restoring new nodes.
Successfully restored new nodes.
Successfully rebuild new nodes.
Configuration is completed.
Starting new nodes.
Successfully started new nodes.
Synchronizing.
Updating cluster configuration.
Successfully updated cluster configuration.
Synchronization is completed.
Successfully enable node whiteList.
Update resource control config file result: Not found resource config file on this node..
Starting new cluster.
Waiting for the cluster status to become available.
.....
The cluster status is Degraded.
Unlocking cluster.
Successfully unlocked cluster.
Creating new node group.
Successfully created new node group.
Waiting for the redistributing status to become yes.
The cluster redistributing status is yes.
Successfully started new cluster.
Successfully disabled delay xlog recycle.
Starting timeseries table redistribute.
Dilatation succeed.
```

Check the node addition progress as user **omm**.

```
# You can run the following command to check the node addition phase:
gs_expand -t progress -X /opt/software/gaussdb/clusterconfig_dilatation.xml
# Before node addition
Querying current dilatation progress.
Current expansion progress is:
Status: Expansion completed or not started, Detail: No exist expansion progress and step file.

# Initialization for node addition
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Init expand

# Installation of new nodes
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Install new nodes

# Node addition completed
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Begin redistribute
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- In case of node addition error during the scale-out, the system automatically rolls back the cluster. If the rollback fails, run the **crashclean** command of the gs_expand tool to clean the cluster and then run the cluster scale-out command described in **Step 7** again.

```
gs_expand -t crashclean -X /opt/software/gaussdb/clusterconfig_dilatation.xml
Distributing cluster dilatation XML file.
Checking static configuration.
Static configuration is matched with the old configuration file.
Crash clean for extansion.
Checking the cluster status.
Last time end with Config new nodes.
Rolling back.
Unlocking cluster.
Successfully unlocked cluster.
Deleting instances from ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Successfully deleted instances from ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Uninstalling applications on ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Successfully uninstalled applications on ['gaussdb04', 'gaussdb05', 'gaussdb06'].
Rollback is completed.
Cleaning  backuped cluster parameter file.
Successfully cleaned backuped cluster parameter file.
Crashclean succeed.
```

- After node addition is successful, the redistribution status changes to **Yes**, preparing for subsequent data redistribution.
- If any new host is disconnected during node addition, the node addition fails and the program and data directory of the faulty host cannot be rolled back. In this case, the cluster is rolled back to the status before scale-out, and you need to manually delete the faulty host and run the scale-out command again.
- If the GUC parameters listen_addresses, local_bind_address, port, pgxc_node_name, pooler_port, log_directory, and audit_directory have been modified before the scale-out, the values of these parameters on the original hosts cannot be synchronized to the new hosts after the scale-out. Instead, the default values are used. In this case, you need to modify the GUC parameters on the new hosts. For details about the parameters, see "Database GUC Parameters > GUC Parameters" in the product documentation.

Step 8 Restore the instances in the cluster to the initial primary/standby status.

```
gs_om -t switch --reset
Operating: Switch reset.
cm_ctl: switchover successfully.
Operation succeeded: Switch reset.
```

Step 9 Set redistribution parameters.

```
# In online scale-out and redistribution scenarios, you are advised to set redis-bucket-batchsize to 10 and redis-bucket-tx-bucket-max to 20.
gs_expand -t parametercfg --redis-bucket-batchsize=10  --redis-bucket-tx-bucket-max=20
Redis parameters are set successfully.
```

Step 10 Perform data redistribution.

```
# parallel-jobs specifies the number of concurrent tasks during data redistribution. The default value is 1. It is advisable that this parameter be set to 4 and you can set it to 8 to the maximum. Set this parameter based on the environment.
gs_expand -t redistribute --redis-mode=insert --parallel-jobs=4
Last time end with Begin redistribute.
Checking the cluster status.
Redistribution
Successfully setup resource control.
If you want to check the progress of hashbucket table redistribution, please check the log file on gaussdb01: /data/lixy1/install/log/lixy/bin/gs_redis/gs_redis_bucket-2025-05-08_144810.log.
Successfully do hashbucket redistribution.
If you want to check the progress of logic redistribution, please check the log file on gaussdb01: /data/lixy1/install/log/lixy/bin/gs_redis/gs_redis-2025-05-08_144816.log.
Successfully clear resource control.
Successfully do common redistribution.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
No need to do timeseries redistribute.
Redistribution succeeded.
```

NOTE

- If the following log is displayed during redistribution, the table is being redistributed. You can go to the *$GAUSSLOG*/**bin/gs_redis/** directory, open the corresponding log, and query the redistribution status of the table.

  ```
  # Hash bucket tables are being redistributed.
  If you want to check the progress of hashbucket table redistribution, please check the log file on gaussdb01: $GAUSSLOG/bin/gs_redis/gs_redis_bucket-2025-05-08_144810.log.
  # Non-hash bucket tables are being redistributed.
  If you want to check the progress of logic redistribution, please check the log file on gaussdb01: $GAUSSLOG/bin/gs_redis/gs_redis-2025-05-08_144816.log.
  ```

- During redistribution, do not query a table for more than 20 minutes because a write lock will be placed if the query is not complete within 20 minutes. Otherwise, the redistribution may fail due to lock timeout.
- After redistribution, ensure that the **default_transaction_isolation** parameter of CNs and DNs has been set to **read committed**. For details about the default_transaction_isolation parameter, see "Database GUC Parameters > GUC Parameters > Default Settings for Client Connections > Statement Behavior" in the Reference.

Step 11 Connect to the default database and check the redistribution status.

```
gaussdb=# select * from redis_status order by updated;
        status          |         updated
------------------------+-------------------------
 REDISTRIBUTION SETUP    | 2025-05-08 17:30:37.161
 REDISTRIBUTION STARTED  | 2025-05-08 17:30:38.899
 REDISTRIBUTION COMPLETE | 2025-05-08 17:37:31.653
 REDISTRIBUTION SHUTDOWN | 2025-05-08 17:37:36.485
(4 rows)
```

You can also run the following command as the cluster user **omm** to check the redistribution progress.

```
# You can run the following command to query the redistribution phase:
gs_expand -t progress -X /opt/software/gaussdb/clusterconfig_dilatation.xml
# Pre-check before redistribution
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Begin redistribute

# Preparation for redistribution
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Prepare redistribute

# Redistribution in progress
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Do redistribute

# Post-processing after redistribution
Querying current dilatation progress.
Current expansion progress is:
Status: Running, Detail: Post redistribute
```

----End

7.26.4 Typical Issues

Q: What should I do if the error message "visibility_heap_tuple_satisfies_mvcc snapshot csn <number> less or equal to bucketcsn <number> from old DN." is displayed during online scale-out?

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

A: This error may be reported when a long transaction snapshot created before the scale-out reads the data that goes online on the new node. This is normal and does not affect service running.

Q: What should I do if the error message "Failed to lock cluster." is displayed during node addition?

A: Check whether there are long transactions that are not terminated in the cluster. If so, the cluster lock cannot be obtained during node addition.

7.27 Read on Standby

7.27.1 Overview

Description

After the read on standby feature is enabled, the standby node supports read-only transactions on the premise of data consistency, effectively relieving the load pressure on the primary node.

Scenarios

In a system that needs to process highly concurrent access requests and requires high reliability and availability, usually, multiple copies are deployed. Standby nodes can be set readable so that you can distribute some query requests to them. This reduces the load on the primary node, fully utilizes the hardware resources of the standby nodes, and reduces hardware investment costs.

Quick Startup

- To configure the read on standby function, set **hot_standby** to **on**; in GTM-lite mode, set **auto_csn_barrier** to **on**.
- Connect to the database using gsql.
  ```
  gsql -d {DB_NAME} -p {DB_PORT} -h {DB_HOST} ...
  ```
  - **DB_NAME**: database name.
  - **DB_PORT**: port of the database CN.
  - **DB_HOST**: host name of the database CN.

  In single-cluster mode or on the CN of the primary cluster deployed in DR mode, after the connection is established, run the following command to enable session-level read on standby:

  ```
  SET enable_standby_read=on;
  ```

  In the standby cluster deployed in DR mode, do not perform this configuration. After the session is established, the DR read function is enabled by default.
- Connect to the database using JDBC.

  Single-cluster configuration:
  ```
  jdbc:gaussdb://node1:port,node2:port,node3:port/database?enableStandbyRead=true
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

enableStandbyRead: enables read on standby after the CN is connected. This parameter takes effect in conjunction with other parameters. For details, see **parameter settings**.

DR read (standby cluster) configuration:

```
jdbc:gaussdb://node1:port,node2:port,node3:port/database?
isasterToleranceCluster=true&autoBalance=true
```

- **disasterToleranceCluster**: enables the system to automatically determine whether the target cluster is the primary cluster or standby cluster. The configured IP address must point to the IP address of the CN in the target cluster.
- **autoBalance**: enables load balancing.
  - If this parameter is set to **true**, **balance**, or **roundrobin**, the JDBC load balancing function is enabled to allocate multiple connections of an application to each CN available in the database cluster.
  - If this parameter is set to **priorityn**, the priority-based JDBC load balancing function is enabled to evenly allocate multiple connections of an application to the first *n* available CNs in the URL.
  - If this parameter is set to **shuffle**, JDBC random load balancing is enabled to randomly and evenly allocate multiple connections of the application to available CNs in the database cluster.
  - If this parameter is set to **specified**, the JDBC load balancing function is enabled and connections are established only on nodes specified in the user URL.
  - If this parameter is set to **shufflen**, JDBC random load balancing is enabled on specified nodes and multiple connections of an application are preferentially and randomly allocated to the first *n* available CNs in the URL. If all the first *n* nodes configured in the URL are unavailable, connections are randomly allocated to other available CNs.
  - If this parameter is set to **false**, the JDBC load balancing and priority-based load balancing functions are disabled.

NOTE

When distributed read on standby is enabled for the standby cluster in DR, **enableStandbyRead** cannot be set to **on**; otherwise, requests will be intercepted.

Specifications and Limitations

- The performance of read on standby is closely related to the hardware specifications, I/O load, and actual services and varies greatly with different service scenarios.
- In typical commercial scenarios (300 GB large table, 100 concurrent UPDATE +VACUUM operations on the primary node, 200 concurrent read operations on the primary and standby nodes, and no bottleneck in I/O and CPU resources):
  - In serial replay mode, the performance of read on standby reaches 90% of that on the primary node.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - In parallel replay mode, the performance of read on standby is 45% to 90% when VACUUM is triggered.
  - In ultimate RTO mode, the performance of read on standby deteriorates by up to 10% (about 80% of that on the primary node) compared with that in serial replay mode.
- Read-only operations do not block log replay of DML operations. The delay between the primary and standby nodes can be controlled within 1s.
- Read-only operations block log replay of DDL operations such as system catalog modification. When a conflict occurs, the query is terminated when the value of **max_standby_streaming_delay** is reached.
- When ultimate RTO is enabled for read-only operations, read-only operations block log replay of some DDL operations but do not affect that of DML operations. Additional memory or disk resources need to be allocated to cache historical data. The query time is subject to the storage space.
- After a switchover, abnormal restart of the standby node, addition of the standby node, or addition of a copy, to enable read on standby, logs must be replayed to the position last read.
- The **enable_standby_read** parameter cannot be enabled in a transaction block or a stored procedure.
- Read on standby does not support autonomous transactions in a distributed system.
- After ultimate RTO is enabled, when a DDL operation is performed to delete files, the deletion on the standby node is delayed. As a result, the number of files in the data directory of the standby node is greater than that of the primary node. In this case, if you query the size of the primary and standby nodes, the size of the standby node is greater than that of the primary node.
- After read on standby in ultimate RTO mode is enabled, the isolation level can only be set to repeatable read. If other isolation levels are set, no error is reported but the settings do not take effect.
- Read on standby does not support temporary tables because temporary tables are not recorded in Xlogs and data is not synchronized to the standby node. Therefore, no temporary tables will be read from the standby node.

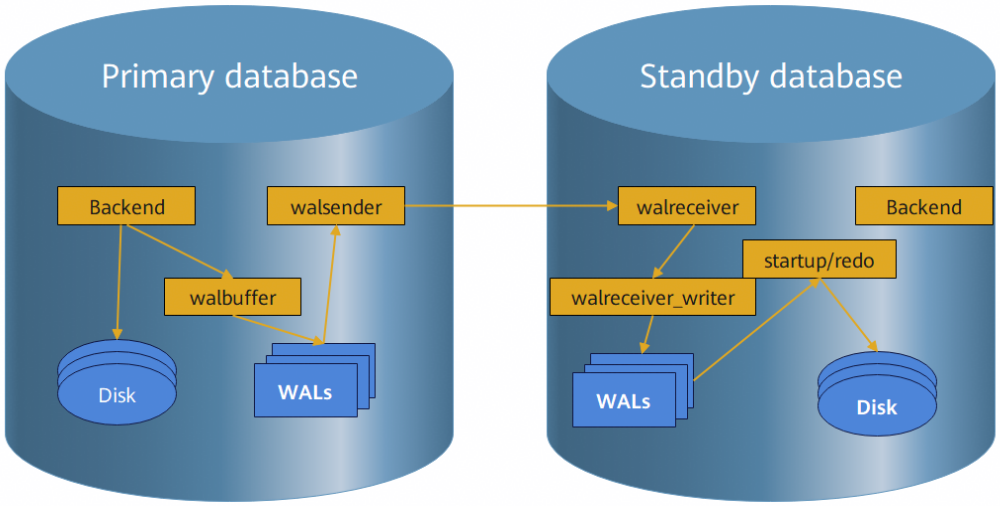
7.27.2 Technical Principle

7.27.2.1 Three Replay Modes of Read on Standby

Redo logs (Xlogs for short) record data modification, which are used to restore data that has been acknowledged but not written into the database. Each GaussDB standby node synchronizes Xlogs and replays them to synchronize data between the primary and standby nodes. **Figure 7-49** shows the streaming synchronization principle.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-49 Streaming synchronization principle



1. A session on the primary node writes Xlogs to disks during data update.
2. The WAL sender thread continuously transmits the Xlogs that have been written to the local disks to the WAL receiver process on the standby node.
3. The WAL writer process on the standby node writes the received Xlogs to disks and reports the receiving position, flushing position, and current replay position to the primary node for majority synchronization decision-making.
4. The replayer thread on the standby node reads local Xlogs and replays them to complete data synchronization.

Serial Replay

- Xlogs are replayed in sequence based on the sequence recorded on the primary node.
- For read on standby in serial replay mode on each CN/DN:

  When the reader thread on the standby node reads a relation for the first time, it obtains the latest global XMIN of the Xlog that has been replayed on the standby node as the local snapshot XMIN, and starts to read Xlogs based on the snapshot.

  The transaction status change can be identified by two Xlog tags: "csn committing" and "csn committed". After the Xlog marked with "csn committed" has been replayed on the standby node, the transaction becomes visible.
- Page clean up (by VACUUM) log conflict:

  When the standby node replays the VACUUM logs, if the log replay conflicts with a read transaction of a small XMIN in the snapshot on the standby node, the read session will be forcibly terminated after the time specified by **max_standby_streaming_delay** expires.

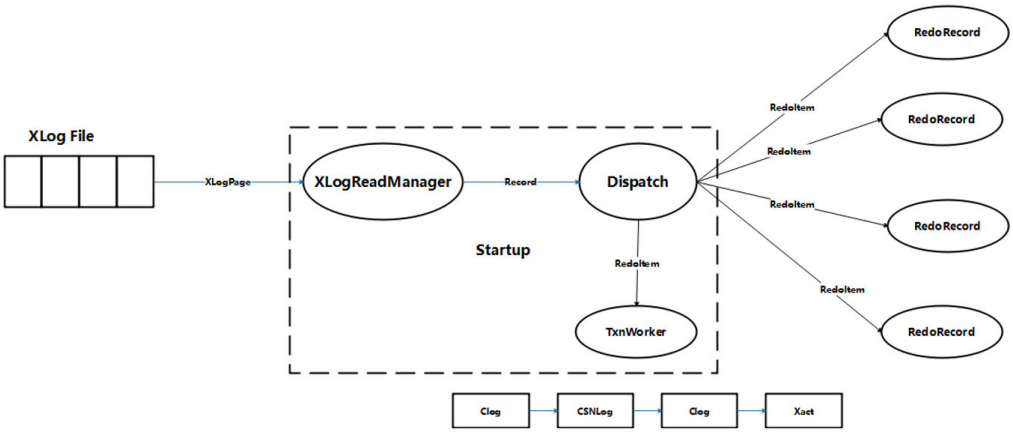
Parallel Replay

The most time-consuming part of the log replay process is allocated to multiple threads by relation for parallel processing. This relies on the multi-core processing

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

capability of modern CPUs. **Figure 7-50** shows the log distribution model in parallel replay mode.

Figure 7-50 Log distribution model in parallel replay mode



- The standby node reads Xlogs in sequence through the WAL reader, and distributes the Xlogs to different redo workers by relation ID for parallel processing.
- The Xlogs committed by transactions are replayed by an independent transaction replayer thread.
- For read on standby in parallel replay mode on each CN/DN:

  The mechanism for resolving conflict between query and replay is the same as that in serial replay mode.
- Parallel replay is implemented at the file level. At a certain time, the following two exceptions may occur due to the difference between the index replay speed and the table replay speed:

  a. Index replay is faster than table replay. Specifically, the heap tuple to which the index tuple points has not been written to the heap.

  b. Index replay is slower than table replay. Specifically, the index tuple pointing to a heap has not been written to the index.

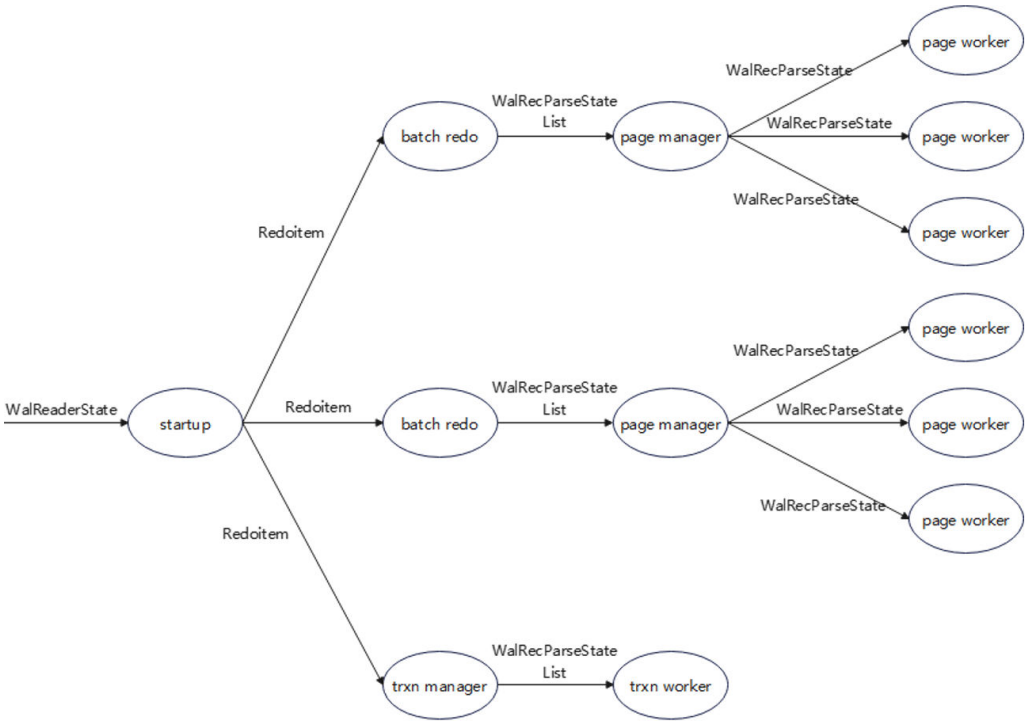
  In case 1, the index scan operator accesses indexes and then the corresponding tables, so that index replay is faster than table replay. An error will be reported when the TID does not exist or the page does not exist.

Replay in Ultimate RTO Mode

By fully utilizing the multi-core processing capability of modern CPUs, the replay process is optimized to a multi-level pipeline architecture, including an independent transaction replay pipeline and multiple data replay pipelines. Xlogs are distributed by page for replay. **Figure 7-51** shows the replay principle in ultimate RTO mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-51 Replay principle in ultimate RTO mode



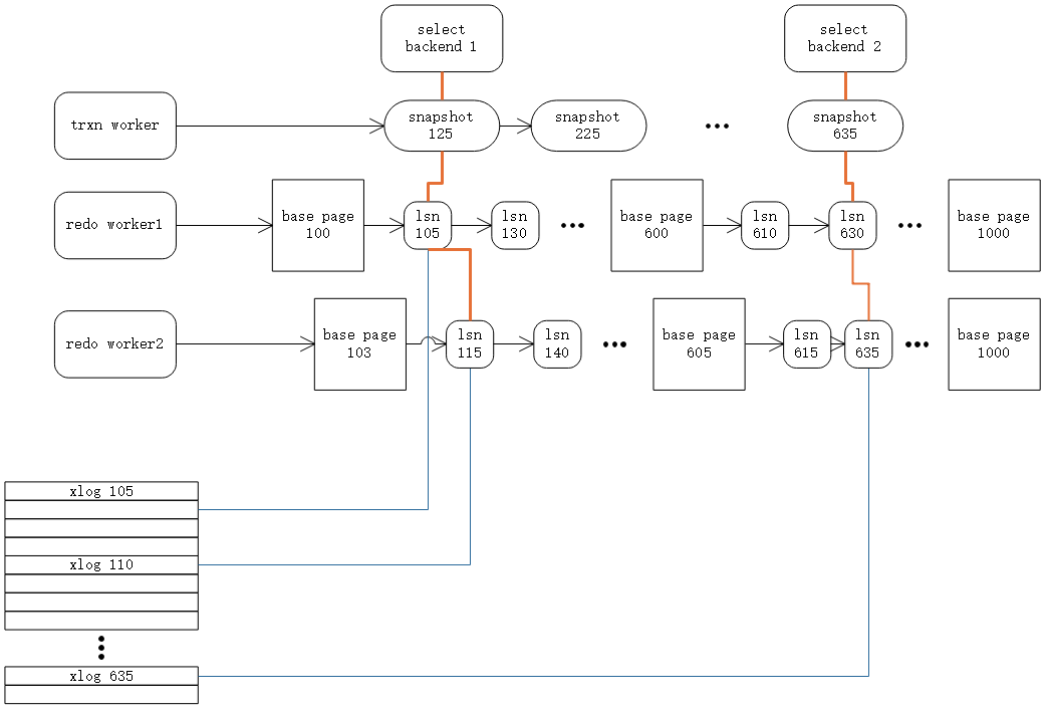
Compared with the parallel replay mode, ultimate RTO replay further improves the parallelism and builds a multi-level pipeline architecture. Each pipeline distributes Xlogs using the page manager to multiple page workers for parallel processing.

For read on standby in ultimate RTO mode on each CN/DN:

Along with the page replay process, historical pages will be overwritten. To address this, multi-version read is introduced to create a base page version and later Xlogs are persistently stored in the LSN chain form. When the pages are read, the page version can be determined based on the snapshot LSN. **Figure 7-52** shows the principle of read on the standby node in ultimate RTO mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-52 Principle of read on the standby node in ultimate RTO mode



When the select backend1 thread attempts to read pages, it reads a relation (for example, relation 1) to obtain the current ultimate RTO snapshot (for example, snapshot 125), extracts the base page 100 of relation 1, and replays Xlogs on the LSN chain until the snapshot LSN read threshold is reached. Then, it repeats the preceding process to read the base pages of other pages (relation 2 and so on) and replays Xlogs on the corresponding LSN chain. In this way, all pages are integrated to complete the read operation on historical versions.

For some DDL operations that delete physical data pages, such as VACUUM, DROP TABLE, TRUNCATE TABLE, and DROP DATABASE, the delayed deletion mechanism is added. In addition, the MVCC mechanism of system catalogs and invalid syscache messages are used to implement multi-version processing of system catalogs, effectively resolving the conflict between query and replay of such DDL operations.

7.27.2.2 Read on Standby in Distributed GTM-Free Mode

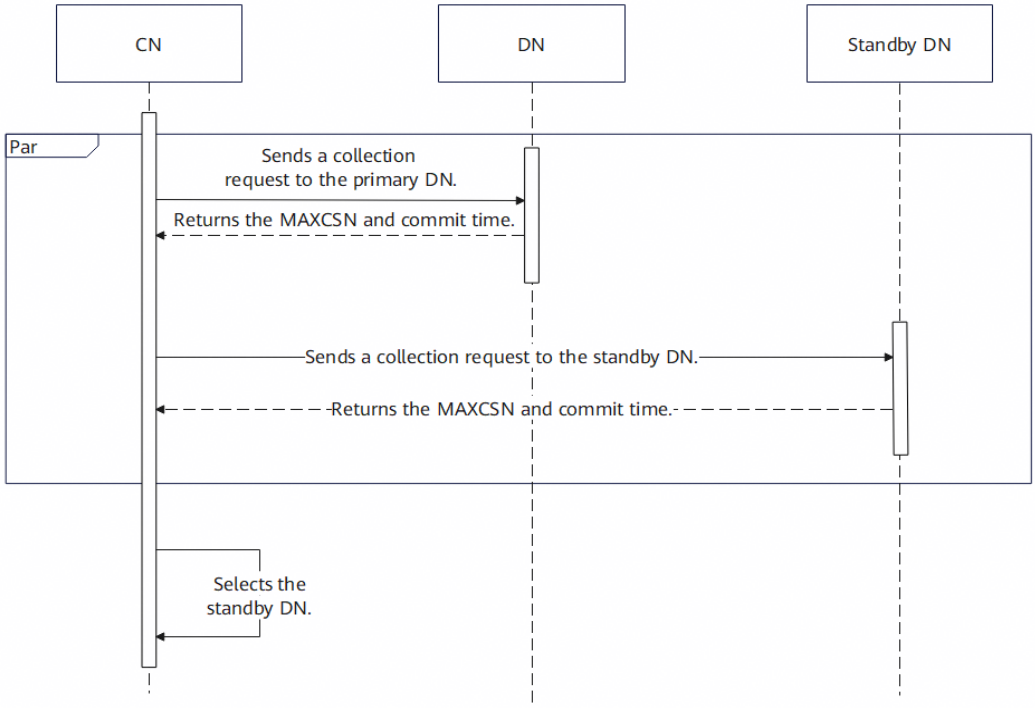
In distributed GTM-free mode, the standby node provides the read capability with weak consistency to effectively reduce the load on the primary node.

In distributed scenarios, the built-in collector thread on the CN continuously collects CSNs and commit time from the primary and standby DNs to check whether the CSNs and transaction delay on the standby DNs meet the preset threshold, so as to deliver read requests to those available standby DNs for execution.

The collector thread on the CN periodically summarizes the MAXCSNs and commit time of each standby DN and records them in the CN cache. **Figure 7-53** shows the process.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

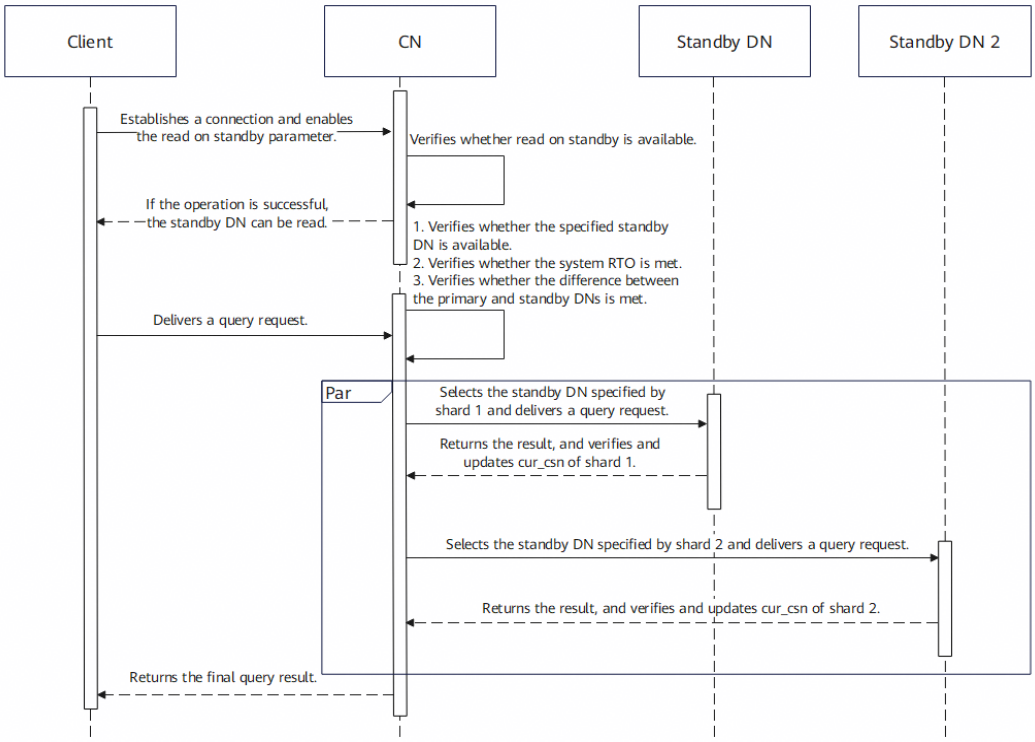
Figure 7-53 Collector thread on the CN



When a query is executed, the latest commit time on a DN must meet the requirements specified by the **standby_read_delay** and **standby_read_rto** parameters and available DNs are independently selected inside each shard. The query request is then delivered to the selected DN for query based on the local snapshot. If no DN is available in all shards, an exception is thrown at the CN layer and the request is terminated.

Figure 7-54 shows the principle of read on the standby node in GTM-free mode.

Figure 7-54 Read on the standby node in GTM-free mode



NOTE

Read on standby in GTM-free mode only checks the delay of standby DNs relative to the primary DN in each shard. It does not ensure that all shards use the same global transaction ID for read. Therefore, instant consistency cannot be achieved. This solution only guarantees eventual consistency. That is, after log replay is complete for all tables involved in the query in all shards, the query result on DNs is the same as that on the primary DN.

Limitations

1. There is a delay between the data read from the standby node and that read from the primary node. If the difference between the data read from the primary and standby nodes exceeds the value specified by **standby_read_delay**, an error is reported.
2. Weak read consistency is provided. However, only snapshots of incremental backups in a session are supported. Read consistency is not guaranteed among sessions.
3. The stream plan is not supported. After the **enable_stream_operator** parameter is enabled, the PG-XC plan is used, rather than the stream plan, to read standby nodes.
4. Load balancing among standby nodes is not supported.
5. The performance and RTO of the primary node take precedence over read on standby. If no standby node is available, an error is reported and the corresponding read on standby requests will not be sent to the primary node.
6. Read on standby affects the RTO of the standby nodes. If the RTO value of all standby nodes exceeds the threshold specified by **standby_read_rto**, an error is reported during read on standby. Service fusing is triggered based on the returned error code.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7. Users can use gsql to directly connect to a standby node in GTM-free mode.

7.27.2.3 Read on Standby in Distributed GTM-Lite Mode

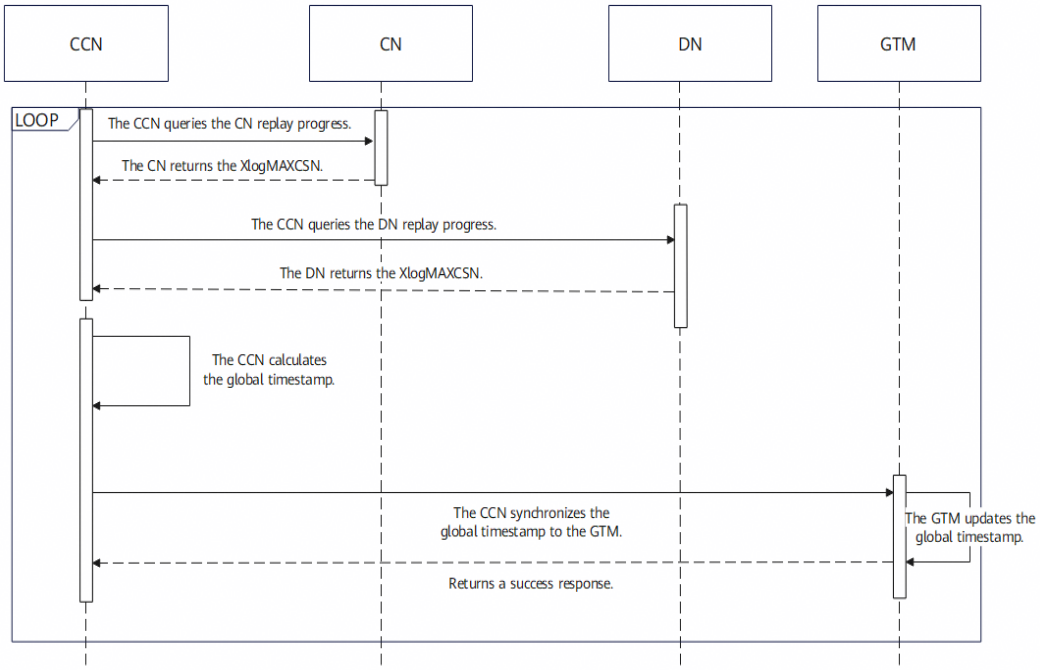
The read capability with strong consistency is provided.

As you may learn from **Three Replay Modes of Read on Standby**, in serial and parallel modes, GaussDB determines the visibility of tuples by comparing the XMINs in tuples and snapshots. If a tuple comes from a transaction whose CSN is less than the CSN in the snapshot, the tuple is visible to the current query. After obtaining a snapshot on the CN in the cluster, the query statement delivers the consistency point to a DN. The DN uses **ConsistencyPointCSN** in the snapshot delivered by the CN to determine the visibility of tuples.

The GUC parameter **auto_csn_barrier** can be enabled to automatically create barriers. The built-in collector thread on the CN continuously collects the CSNs in the barriers from standby DNs, calculates the CSN for global consistency, and persistently stores and caches information of the standby DNs. The query thread filters the standby DNs that meet the consistency point based on the cached data and delivers some read requests to the target standby DN for execution. The standby DN reads the consistency point to perform the query operation, and returns the query result to the CN.

Figure 7-55 shows the process of enabling automatic barrier creation for CSNs on the CN and recording the CSNs in the GTM.

Figure 7-55 CSN recording on the CN



1. The GUC parameter **auto_csn_barrier** is set to **on** to enable automatic barrier creation and generate barrier logs. When each standby DN is replayed, the value of **XlogMaxCSN** is updated.
2. The background thread standby_read_main (reusing the background thread in GTM-free mode) on the CN periodically maintains the global consistency

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

point. This thread periodically sends messages to all standby DNs to query the log replay progress.

3. After receiving the query request from the CN, each standby DN returns the maximum CSN of the barrier log replayed by the standby DN, that is, **XlogMaxCSN**.
4. The CN calculates the median of **XlogMaxCSN** of each shard, uses the minimum median value of all shards as the consistency point, and stores the CSN to the GTM.

This mechanism ensures that Xlogs whose CSNs are smaller than the consistency point have been replayed on most nodes. After the collection is complete, the CN synchronizes the consistency point to the GTM for query.

Ultimate RTO Scenario

For read on standby in ultimate RTO mode, the **read_lsn** field is added to the snapshot to obtain the historical page data of the specified LSN version. DNs need to maintain the mapping between CSNs and LSNs and record XMIN/XMAX information. When a DN receives a snapshot containing the global consistency point delivered by a CN, the DN searches for the corresponding LSN and XMIN and XMAX information based on the CSN to construct a complete snapshot. The following figure shows the CSN-LSN information structure.

Figure 7-56 CSN-LSN information structure

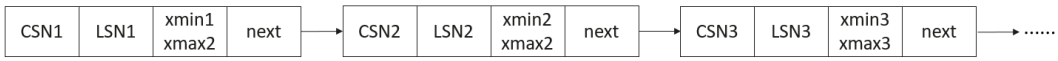
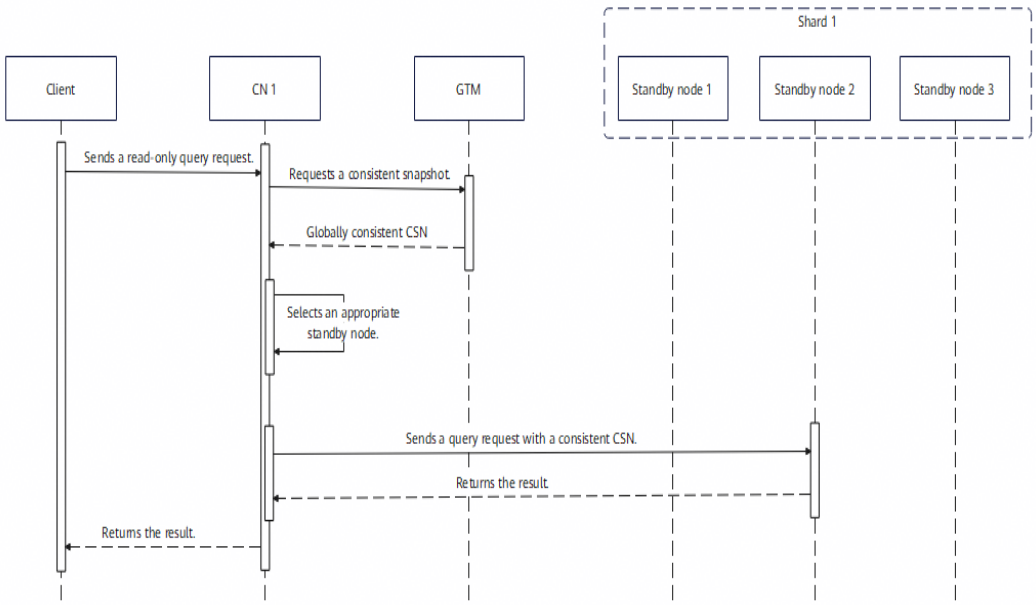


Figure 7-57 Overall query process



The query process is as follows:

1. After the **enable_standby_read** parameter is enabled on the client, the client sends a query request to the CN.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. The CN obtains the consistency point set by the background thread standby_read_main from the GTM and injects it to the local snapshot.
3. The CN selects a standby DN that meets the conditions based on the snapshot CSN. (In ultimate RTO mode, there is an additional verification on the recycling position, that is, whether the corresponding snapshot is stored on the CSN-LSN information chain on the DN).
4. The CN delivers a query request to the selected standby DN.
5. After receiving the request from the CN, the standby DN injects the received CSN into the local snapshot to ensure that the CN and DN use the same global CSN for query.
6. The standby DN executes the query and returns the result to the CN.
7. The CN returns the query result to the client.

Limitations

- Specifications and limitations of read on standby in distributed GTM-lite mode:
  - Prerequisites: The barrier function is enabled. This function is controlled by the **auto_csn_barrier** parameter.
  - Limitations regarding conflict between replay and query
    - When a lock-related query (mainly DDL operations on tables) conflicts with serial/parallel replay, similar to that conflict with read on standby, the query ends depending on the setting of **max_standby_streaming_delay** and reporting the following error: "canceling statement due to conflict with recovery." For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Standby Server" in *Reference*.
    - The query time exceeds the value of the **standby_max_query_time** parameter and reports the following error: "User query has exceeded the duration threshold." For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Standby Server" in *Reference*.
    - Forcible recycling of files read by the standby node is triggered, which is controlled by the **max_standby_base_page_size**, **max_standby_lsn_info_size**, and **standby_force_recycle_ratio** parameters. For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Standby Server" in *Reference*.
    - Logs related to segment-page-based physical space shrinking are replayed on the standby node.
    - When **REINDEX DATABASE** is executed on the primary node, the query on the standby node conflicts with replay of relmap logs and reports the following error message: "User query might have needed to see row versions that must be removed."
    - When **DROP DATABASE** or **DROP TABLESPACE** is executed on the primary node, the query on the standby node may conflict with log replay and reports the following error: "canceling statement due to conflict with recovery."

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Snapshot limitations
  - In scenarios where data is read from the standby node with ultimate RTO, snapshot information is sent by the CN, which may be delayed. Probably, the data information has been updated. Errors may be reported in the following scenarios:
    - Operations such as REINDEX DATABASE are performed, causing relmap changes, and the error message "snapshot lsn is smaller than lastReplayedConflictLsn due to relmap redo" is displayed.
    - When the node is started, the error message "snapshot lsn is smaller than lastReplayedConflictLsn due to starting action" is displayed.
    - When the files read from the standby node are forcibly recycled, the error message "snapshot lsn is smaller than lastReplayedConflictLsn due to force recycle" is displayed.
  - In the read on standby with ultimate RTO scenario, forcible recycling is triggered due to cluster startup, shutdown, or fault, or due to a certain reason (such as insufficient disk space, GUC parameter modification, or replay failure on some nodes) in ultimate RTO mode, the standby node may report the following read error and end the query: "ERROR: snapshot csn is not in csn-lsn list, DETAL: snapshot csn is 123456789."
  - In serial/parallel replay scenarios where data is read from the standby node, if VACUUM is frequently performed on the primary node, the data read from the standby node will be cleared in advance. As a result, a conflict may occur between the query and replay. The standby node may report the following read error and end the query: "ERROR: gtm csn small: gtm csn 123456789, lastReplayedConflictCSN 123456999."
  - When DDL operations are performed on a table, the system catalog on the CN has been modified, but the standby DN may not replay these DDL logs. When the old snapshot is used to query the table, the metadata on the CN may be inconsistent with that on the standby DN. After detecting this situation, the query on the table will end and the following error is reported: "ERROR: current snapshot is invalid for this relation/partition." In this case, retry is required.
  - When connecting to DNs, the CN needs to calculate the available DN copies in the corresponding shard based on the current global consistency point. The following error may occur in some scenarios:

    ```
    LOG: Current slice cannot select a normal standby for standby read, slice: <int>  ...
    ERROR: pooler_get_dn_handles invalid Datanode number: -1, dnNum: <int>, u_sess->pgxc_cxt.NumDataNodes[<int>]
    ```

    - Scenario 1: The replay speed of the same data copy on two DNs is different, and the DN that replays data faster restarts.

      Solution: Wait until the slower shard catches up with the global consistency point. Then, the error will be automatically cleared.
    - Scenario 2: DR mapping is just constructed, but the replay and information collection of each CN and DN have not been completed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Solution: This error occurs only in the initial phase of the startup and will be automatically cleared after several seconds.

- Scenario 3: In the ultimate RTO mode, the replay progress of different shards varies greatly. As a result, the recycling point of the shard with faster replay exceeds the global consistency point.

  Solution: Wait until the service peak ends. After the replay differences among shards are reduced, the error is cleared. If the replay of only a few CNs is slow, run the **cm_ctl stop** command to manually stop them.

– Resource control

  - Disk space: Currently, the space usage for read on standby in ultimate RTO mode is controlled by the **max_standby_base_page_size**, **max_standby_lsn_info_size**, and **standby_force_recycle_ratio** parameters. For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Standby Server" in *Reference*.
  - Memory and I/O: Independent buffer pool (controlled by the **enable_standby_bufferpool** parameter) and dirty page flushing are supported for read on standby. For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Standby Server" in *Reference*.
  - CPU: Currently, there is no resource control capability. When the CPU resources of the node are sufficient (CPU resource usage ≤ 70%), the read on standby and replay are not affected.
  - In scenarios of low specifications (8 cores or less), ultimate RTO replay occupies a large number of resources. Therefore, you are advised not to enable read on standby in ultimate RTO mode.

– Performance specifications

  - If the AZ selection parameter **standby_read_use_az_info** or load balance parameter **standby_read_use_load_balance** is enabled, all available standby DNs need to be traversed for selection. As a result, the query performance deteriorates.
  - When DDL operations are frequently performed on the primary node, the query on the standby node may be slow or even canceled due to conflict with replay.
  - In the 96-core 768 GB physical machine environment where 3-CN 3-DN hybrid deployment is used, independent data disks and egress parameters are configured, if the initial data volume is 100 GB, 100 concurrent read operations are performed on the primary and standby nodes, 100 concurrent update operations are performed on the primary node, and 100 concurrent insert operations are performed on the primary node, the read performance of the standby node reaches 80% of that of the primary node.

– When the primary node is rebuilding indexes online, dropping indexes, rebuilding indexes, or dropping partitions, the query delivered by the CN to the standby node may access the indexes that have been dropped. In this case, the following error may be reported and retry is required:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - For an ordinary table: "Could not open relation with OID xxx."
  - For a partitioned table: "Partition oid xxx is invalid when opening partition current tid."
  - "could not open block during recovery delete object, please try again later."
  - "search sys cache failed when get index list, indexrelid xxx, indrelid xxx, relation xxx."
  - "partition xxx does not exist on relation xxx."
- When the primary node concurrently executes addition and deletion statements (such as ALTER TABLE ADD FIRST/AFTER) at a specified location, causing concurrent modifications to the system catalog, an error indicating that the column type does not match may occur during the query on the standby node. In this case, you need to retry the service.
- In distributed scenarios, when the primary node executes DDL statements and the standby node executes the stream plan for query, the table definition displayed on the CN may be inconsistent with that on the standby DN. As a result, an error is reported during the query. The reason is as follows: If read on standby is enabled in distributed deployment and a stream plan is used for query, the plan is generated on the CN and then delivered to the standby DN for execution. However, the CN, as the primary node, uses SnapshotNow to access the system catalog when generating the stream plan. When the standby DN executes the plan, SnapshotMvcc is used to read data. Therefore, the data is inconsistent with the table definition in the plan, causing the query error. This problem is not involved in the distributed DR read scenario because the CN of the standby cluster is also a standby node and uses SnapshotMvcc to access the system catalog. The DDL operations include adding or dropping columns, renaming columns, extending the size of varchar columns, modifying the precision of numeric columns, modifying ENUM/SET column definitions, renaming tables, adding or dropping partitions, renaming tablespaces, creating indexes CONCURRENTLY, and rebuilding indexes CONCURRENTLY. When this type of error occurs, retry read on the standby node. The main errors that may be reported are as follows:

```
attribute xxx has wrong type
invalid attribute number xxx
attribute number xxx exceeds number of columns xxx
pg_class entry for relid xxx vanished during xxx
cache lookup failed for index xxx
```

7.27.2.4 DR Read in Distributed GTM-Lite Mode

The implementation principle of distributed DR read in GTM-lite mode is basically the same as that of distributed read on standby.

After the **auto_csn_barrier** parameter is enabled, the system performs automatic barrier creation. In the DR read scenario, the collector thread continuously collects the CSNs in the barriers from standby DNs and sends a request to each CN, including itself. Each CN is considered as an independent shard for verification. This mechanism requires that at least one CN can obtain a valid CSN so as to calculate a globally consistent CSN and save it to the GTM.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

In the query execution phase, when all standby DNs meet the consistency point, the system preferentially selects the cascaded standby DN to deliver the read request. The cascaded standby DN obtains the consistency point of read on standby, queries the record corresponding to the LSN in the CSN-LSN information, and returns the query result to the CN.

Distributed read depends on the node information maintained by the pgxc_node system catalog to initialize the local CN communication list. In the DR scenario, the CN of the standby cluster is the standby node of the CN in the primary cluster, which is built by replaying the data in the pgxc_node system catalog of the primary cluster and cannot update its content in the table by itself. Therefore, the pgxc_node system catalog of the primary cluster needs to record the static information about all nodes in the standby cluster. When the standby cluster accesses the pgxc_node system catalog, it matches the **cluster_name** column in the table based on the value of GUC parameter **cluster_name** (name of the local cluster) to obtain its node information.

Limitations

- DR limitations:
  - The standby cluster does not support the COPY TO operation. This is because an XID is required for COPY TO on a CN, while the standby cluster contains only a standby CN and no XID can be allocated to the standby CN. The following error will be reported for such operation: "ERROR: cannot assign TransactionIds during recovery."
  - If there is only one copy in distributed DR mode, distributed read operations are not supported. You can perform a local query in maintenance mode. The following error message will be displayed when a distributed query is executed: "ERROR: Currently disaster cluster read is not supported."
  - If the primary cluster is disconnected from the standby cluster, the data queried from the standby cluster may be old data.
  - In the GTM-lite solution, when a CN is connected to DNs, it calculates the available DN copy in the corresponding shard based on the current global consistency point. In this case, the error "Current slice cannot select a normal standby for standby read, slice: 1" may be reported in the following scenarios:
    - For the same copy, the replay speeds of two DNs are inconsistent and the faster DN restarts. Solution: The global consistency point will not be updated. Wait until the slower shard catches up with the global consistency point. Then, the error disappears.
    - The DR mapping is just constructed, but the replay and recycling information of each CN and DN has not been collected. Solution: No action is required. The startup time window disappears several seconds later.
    - In ultimate RTO mode, the replay speeds vary greatly among shards and the faster shard is recycled ahead of the global consistency point. Solution: Wait until the service peak ends. After the replay differences among shards are reduced, the error is cleared. If the replay of only a few CNs is slow, run the **cm_ctl stop** command to manually stop them.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - When a CN is removed from the standby cluster, a CN is added back, or the version is upgraded to 506 and the upgrade is committed, it affects the communication cache of the pgxc_node table. At this point, the internal reload mechanism is triggered. During the reload period (approximately 3s), the following known errors may occur:
    - "node id %d is in pgxc_group but not in the handles, maybe rebuilding DisasterCache." Root cause: The OID recorded in pgxc_group in the cache does not match that recorded in the communication DN matrix in the cache.
    - "cache lookup failed for node %u." Root cause: The OID in the pgxc_node table does not match the node OID recorded in the cache.
    - "Cannot reload pool when in a transaction block." Root cause: The pgxc_node is updated during a session. However, the transaction does not support reload pool. Therefore, an error is reported, but the error does not persist.
    - pooler: shmemNums does not match with usessNums, pgxc_node reload failed. Root cause: The number of standby nodes recorded in the communication cache agent is old, but the number of standby nodes recorded in the session memory variable is updated with the pgxc_node table.
  - Temporary tables, temporary views, temporary sequences, and unlogged tables do not generate Xlogs and are not synchronized to the standby node. Therefore, the standby node does not support query of these objects. If you query these objects, the following error will be reported:
    - "ERROR: Temporary or unlogged table cannot be accessed on the standby."
- Limitations regarding conflicts between replay and query:
  - Frequent vacuum and DDL operations on the primary cluster affect the log replay speed. The replay may conflict with query. During log replay, if the conflicting session has been executed for a period longer than the time specified by **max_standby_streaming_delay**, the query ends and the following error will be reported: "ERROR: canceling statement due to conflict with recovery" or "ERROR: terminating connection due to conflict with recovery."
- Snapshot limitations:

  DR read during serial/parallel replay:
  - When the standby cluster for DR replays logs of DDL operations performed on the primary cluster and the logs are not replayed successfully on some nodes of the standby cluster, the snapshot verification fails and the query ends reporting the following error: "ERROR:current snapshot is invalid for this partition/relation."
  - If the primary cluster is frequently vacuumed, the data queried by the standby cluster for DR will be cleared in advance. A read session will check whether the snapshot ID is greater than the latest point where a replay and query conflict occurs on the DN. If not, the snapshot is too old and the latest data cannot be read. In this case, the query ends reporting the following error: "ERROR: gtm csn small: gtm csn 123456789, lastReplayedConflictCSN 123456999."

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

DR read in ultimate RTO scenario:

- When the disk space is insufficient, forcible recycling is triggered by modifying GUC parameters, a database startup/shutdown occurs, or a shard replay is very slow, the global consistency point may not be updated promptly. As a result, the CN or DN that is normally replayed cannot be read because the local historical version does not support the current consistency point and the query ends reporting the following error: "ERROR: snapshot csn is not in csn-lsn list."
- System catalogs (**relfilenode** < 16384) do not support delayed DDL operations. When the TRUNCATE or DROP operation is performed on a system catalog, related files, including the historical versions for ultimate RTO, will be directly deleted. In this case, the standby node may report the error message "ERROR: standby_read_buf can't find history version page/ERROR: block old version can not found" during read on standby because the historical versions are not traceable.
- DDL operations are not postponed for segment-page tables. Therefore, when DDL operations such as TRUNCATE, DROP, and REINDEX are performed on segment-page tables, the original data files are directly deleted during replay on the standby node. As a result, an error is reported when segment-page table page 0 is read.

- Performance limitations
  - When the resources that can be occupied by the primary and standby clusters are the same, for the Astore table in parallel replay mode:
    - In the scenario where the primary cluster has no write service and the logs of the standby cluster have been replayed, the query performance of the standby cluster reaches 80% of that of the primary cluster in terms of latency when the SELECT COUNT(*) FROM XXX statement is used.
    - In scenarios with write services, the query performance of the primary cluster and standby cluster may fluctuate due to service impact. A single query does not ensure that the performance of the standby cluster reaches 80% of that of the primary cluster. In the scenario without DDL operations, the performance of the standby cluster reaches 80% of that of the primary cluster in terms of TPS, 95th percentile, and average latency for sysbench and TPC-C loads.
    - If multiple primary or standby nodes to be queried are deployed on the same host and the query performance deteriorates due to insufficient resources on the host, the preceding two scenarios are not met.
    - If the first two scenarios are not met, you are advised to check whether resources (CPU, memory, and disk) on each node in the standby cluster are limited.
    - The Astore table in parallel replay mode on the standby cluster does not support index-only scan query. Therefore, more data pages may need to be read when the SELECT COUNT(*) FROM XXX statement is used. You are advised to adjust the **shared_buffers** parameter to optimize the cache.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Limitations on compatibility scenarios

  During the observation period of upgrade to version 506.0.0, the mapping solution for the source version is used to process the DR read communication list. At this time, the limitations are as follows:

  - The content of the pgxc_node table in the standby cluster for DR is synchronized from the primary cluster. Therefore, the CN in the standby cluster maps to the actual node based on the value of **node_oid** in the pgxc_node table. However, the node names of the primary and standby clusters do not exactly map to each other. As a result, the query result of the view implemented by running **EXECUTEDIRECT ON** in the standby cluster is abnormal (duplicate or missing). You are advised to query the view on the corresponding node.
  - If the primary and standby clusters are heterogeneous and the number of CNs is inconsistent, execute **SELECT * FROM pgxc_disaster_read_status()** to query the standby cluster CNs that are mapped to the primary cluster before reading data from the standby cluster. Then, perform service operations based on the queried CNs. If a CN that does not have a mapping relationship with the primary cluster is connected, the following error may be reported because the CN is not involved in the calculation of global consistency point: "ERROR: cn data invisible: local csn 123456789, gtm snapshotcsn 123456999."
  - If JDBC is used to connect to the standby cluster and the load balancing function is enabled, query requests are sent to the primary cluster. As a result, the query performance is affected.
  - After the CN DR relationship is established, it reads the topology node information of the local cluster, relying on the internal mapping mechanism constructed after the pgxc_node table is replayed. Before the DR mapping is constructed, an error message "Not support disaster read before init." appears for the read request. After the DR relationship is established and the standby cluster is started properly, the read error disappears about 10 seconds later (there is no RTO difference between the primary and standby clusters).
  - Since the standby cluster replays the topology information of the primary cluster recorded in the pgxc_node table, the following errors may be reported in the time window (about 3s) when a new OID is read from the pgxc_node table but the OID (serving as the key for CN query in a distributed system) in the system cache is not updated yet if the pgxc_node table in the standby cluster is updated along with the primary cluster topology change due to (1) a switchover between the primary and standby clusters, (2) a switchover in the 1:*N* primary/standby deployment, or (3) node replacement in or CN removal from the primary cluster:

    - "node id %d is in pgxc_group but not in the handles, maybe rebuilding DisasterCache." Root cause: The OID recorded in pgxc_group in the cache does not match that recorded in the communication DN matrix in the cache.
    - "cache lookup failed for node %u." Root cause: The OID in the pgxc_node table does not match the node OID recorded in the cache.
    - "Cannot reload pool when in a transaction block." Root cause: The pgxc_node is updated during a session. However, the transaction

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

does not support reload pool. Therefore, an error is reported, but the error does not persist.

- pooler: shmemNums does not match with usessNums, pgxc_node reload failed. Root cause: The number of standby nodes recorded in the communication cache agent is old, but the number of standby nodes recorded in the session memory variable is updated with the pgxc_node table.

- Limitations of read on the standby in GTM-lite mode

  The GTM-lite mode uses the same read logic for read in DR and read on standby. For details, see **Specifications and limitations of read on standby in distributed GTM-lite mode**.

7.27.3 Operation Guide

7.27.3.1 Selecting a Replay Mode Based on Your Environment Specifications

Selecting a Replay Mode

To select the serial replay mode, set **recovery_parse_workers** to **1**, **recovery_redo_workers** to **1**, and **recovery_max_workers** to **1**.

To select the parallel replay mode, set **recovery_parse_workers** to **1**, **recovery_redo_workers** to **1**, and **recovery_max_workers** to a value greater than 1.

To select the ultimate RTO replay mode, set **recovery_parse_workers** or **recovery_redo_workers** to a value greater than 1. In this mode, the **recovery_max_workers** parameter is invalid.

By default, read on standby is enabled (**hot_standby** is set to **on**). Read on standby in ultimate RTO mode requires extra disk space to store historical data. The space required is closely related to the actual services. In small-capacity scenarios with limited disk space, you are advised to test and verify the actual service model and batch processing peak characteristics in advance to determine whether the ultimate RTO replay mode is applicable.

In RTO-sensitive scenarios where read on standby is unnecessary, you are advised to disable the read on standby feature to release more CPU and I/O resources for replay on standby.

The recommended replay modes for different hardware specifications are as follows.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-38 Parameter settings for different CPUs, memory sizes, and deployment modes

No.	Number of CPUs	Memory (GB)	Hybrid Deployment or Not	recovery_parse_workers	recovery_redo_workers	Number of Replayer Threads	Remarks
1	4	-	-	1	1	Value of **recovery_max_workers**	Ultimate RTO is not recommended.
2	8	-	Yes	1	1	Value of **recovery_max_workers**	Ultimate RTO is not recommended.
3	8	64	No	1	3	10	-
4	16	128	Yes	1	2	9	-
5	16	128	No	2	3	15	-
6	32	256	Yes	2	2	13	-
7	32	256	No	2	8	25	-
8	64	512	Yes	2	4	17	-
9	64	512	No	2	8	25	Systems with larger hardware specifications should also use this recommended value.
10	96	768	-	2	8	25	Systems with larger hardware specifications should also use this recommended value.

Typical Scenario - 16 Cores

Scenario:

1. Server hardware specifications:

   CPU: 2.60 GHz 16 cores; memory: 4 GB x 16; disk: SSD.
2. sysbench: 200 parallel INSERT+UPDATE (1:1) operations on 5 ordinary tables.
3. Parallel replay GUC parameters: **recovery_parse_workers** set to **1**, **recovery_redo_workers** set to **1**, and **recovery_max_workers** set to **4**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4. Ultimate RTO GUC parameters: **recovery_parse_workers** set to **2**, **recovery_redo_workers** set to **2**, and **recovery_max_workers** set to **4**.

Testing:

Assuming that INSERT and UPDATE (1:1) operations are performed on a specified number (1, 5, 10, and 20) of tables with the size of each table fixed to 20 GB in the sysbench test, compare the replay speed differences between ultimate RTO replay and parallel replay.

Number of Tables	Parallel Replay (MB)	Ultimate RTO (MB)	Difference
1	57	57	0.00%
5	60	64	6.67%
10	47.5	48	1.05%
20	25	30	20.00%

Typical Scenario - 64 Cores

Scenario:

1. Server hardware specifications:

   CPU: Intel(R) Xeon(R) Gold 6278C 2.60 GHz 64 cores; memory: 16 GB x 16; disk: SSD.

2. sysbench: 1000 parallel INSERT+UPDATE (1:1) operations on 5 ordinary tables.
3. Parallel replay GUC parameters: **recovery_parse_workers** set to **1**, **recovery_redo_workers** set to **1**, and **recovery_max_workers** set to **8**.
4. Ultimate RTO GUC parameters: **recovery_parse_workers** set to **2**, **recovery_redo_workers** set to **8**, and **recovery_max_workers** set to **8**.

Testing:

Assuming that long transactions varying in duration (non-long, 6 minutes, and 10 minutes) perform parallel INSERT and UPDATE (1:1) operations on tables in the sysbench test, compare the replay speed of ultimate RTO replay and parallel replay. When long transactions exist, the replay speed of ultimate RTO replay is 26% higher than that of parallel replay.

Long Transaction	Parallel Replay (MB)	Ultimate RTO (MB)	Difference
No long transaction	332.514	343.773	3.39%
6-minute long transaction	288.424	363.432	26.01%

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Long Transaction	Parallel Replay (MB)	Ultimate RTO (MB)	Difference
10-minute long transaction	310.269	399.501	28.76%

Replay Rate in Typical Scenarios

Hardware specifications:

CPU: Intel(R) Xeon(R) Platinum 8378A CPU 3.00 GHz 64 cores; memory: 16 GB x 32; disk: SSD.

Scenario: Assume that 50 parallel INSERT operations are performed on a single ordinary table.

Table 7-39 Comparison of replay speed in typical scenarios

Scenario	Specifications	Replay Type	Parameter Setting	Number of Replayer Threads	Size of Stacked Xlogs	Xlog Generation Speed on the Primary Node	Replay Rate on the Standby Node	buffer_pool
Live-network environment	Primary and standby nodes: 64 cores, 512 GB	Parallel replay	recovery_parse_workers=1 recovery_redo_workers=1 recovery_max_workers=4	4	40 GB	About 240 MB/s	About 120 MB/s	180 GB
Lab environment	Primary and standby nodes: 96 cores, 512 GB	Parallel replay	recovery_parse_workers=1 recovery_redo_workers=1 recovery_max_workers=4 recovery_time_target=3600	4	30 GB	250 MB/s	About 100 MB/s	180 GB

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenario	Specifications	Replay Type	Parameter Setting	Number of Replayer Threads	Size of Stacked Xlogs	Xlog Generation Speed on the Primary Node	Replay Rate on the Standby Node	buffer_pool
Lab environment	Primary and standby nodes: 96 cores, 512 GB	Ultimate RTO	recovery_parse_workers=2 recovery_redo_workers=4 recovery_max_workers=4 recovery_time_target=3600	17	0	250 MB/s	200 MB/s to 250 MB/s	180 GB

7.27.3.2 Procedure

Step 1 Configure kernel-related parameters.

Set **recovery_parse_workers**, **recovery_redo_workers**, and **recovery_max_workers** by referring to **Selecting a Replay Mode Based on Your Environment Specifications**.

```
wal_level=hot_standby
```

The log replay level must be **hot_standby** or higher.

```
hot_standby=on
```

The **read on standby** function must be set to **on**.

```
max_standby_streaming_delay=10s
```

This parameter specifies the time for the standby node to wait before canceling the query when a replay and query conflict occurs. It affects the RTO.

If replay in ultimate RTO mode is configured based on the environment specifications, you need to set the following GUC parameters:

```
standby_max_query_time=10s
```

When recycling points are calculated, the queries that exceed the time specified by **standby_max_query_time** will be canceled. You are advised to set this parameter to the value of **max_standby_streaming_delay**.

```
max_standby_base_page_size=256GB
```

This parameter specifies the threshold of space storing historical page versions for read on standby in ultimate RTO mode. When the space occupied by historical base pages reaches 80% of the threshold, forcible recycling is triggered and all queries that hold snapshots earlier than the new recycling point are canceled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> In ultimate RTO mode, the multi-version read mechanism is introduced to resolve the file deletion replay and query conflict. Therefore, earlier versions of pages need to be continuously saved. No recommended value is provided for the space because it is closely related to the Xlog generation speed and actual services. You can set it by referring to the following suggestions:
>
> 1. When the disk space is limited and long queries on the standby node cannot be avoided, if no error is reported during the read operation and temporary increase of the RTO during some queries is acceptable, you are advised to specify standby nodes to perform read on standby in serial or parallel replay mode.
> 2. When the disk space is sufficient and long queries on the standby node cannot be avoided, if no error is reported during the read operation and increase of the RTO is not acceptable, you are advised to restrict DDL operations in the primary cluster by referring to the ultimate RTO replay and query conflict whitelist, and set **max_standby_base_page_size** and **max_standby_lsn_info_size** to large values based on the Xlog generation speed and **standby_read** directory space bloat ratio. (The empirical value is that the base page size is 5 to 7 times that of **lsn_info**.)
> 3. If read errors can be retried or long queries can be avoided, but there are RTO and disk space requirements, configure the space based on the actual Xlog generation speed and **standby_read** directory space bloat ratio.

```
max_standby_lsn_info_size=256GB
```

This parameter specifies the threshold of space storing historical page versions for read on standby in ultimate RTO mode. When the space occupied by historical base pages reaches 80% of the threshold, forcible recycling is triggered and all queries that hold snapshots earlier than the new recycling point are canceled. You are advised to set it to the value of **max_standby_base_page_size**.

Step 2 Select the GTM-free/GTM-lite mode.

- GTM-free mode:

  ```
  enable_standby_read=on
  ```

  This is a SIGHUP-level GUC parameter. You are advised to set this parameter in the connection string or run the **set enable_standby_read=on;** command after each session is initialized.

  ```
  standby_read_delay=100000000ms
  ```

  This is a SIGHUP-level GUC parameter, which defines the maximum delay between the primary and standby nodes. If the delay exceeds the value of this parameter, requests cannot be delivered to the standby node.

  ```
  standby_read_rto=60000ms
  ```

  Sets the system RTO threshold. If the actual value exceeds the threshold, read on standby is prohibited.

  ```
  enable_gtm_free=on
  ```

  Enable the GTM-free mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

  ```
  gtm_option=2
  ```

  Enable the GTM-free mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

- GTM-lite mode:

  ```
  enable_standby_read=on
  ```

  This is a SIGHUP-level GUC parameter. You are advised to set this parameter in the connection string or run the **set enable_standby_read=on;** command after each session is initialized.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
enable_gtm_free=off
```

Disable the GTM-free mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

```
gtm_option=1
```

Enable the GTM-lite mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

```
auto_csn_barrier=on
```

Enable the automatic barrier function.

- DR read in GTM-lite mode:

```
enable_gtm_free=off
```

Disable the GTM-free mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

```
gtm_option=1
```

Enable the GTM-lite mode. This is a POSTMASTER-level parameter and can take effect only after a restart.

Primary cluster configuration:

```
auto_csn_barrier=on
```

Enable the automatic barrier function.

Step 3 Configure JDBC-related parameters.

Single-cluster configuration:

```
jdbc:gaussdb://node1:port,node2:port,node3:port/database?enableStandbyRead=true
```

enableStandbyRead: enables read on standby after the CN is connected. This parameter takes effect only when it is configured with kernel parameters.

DR read (standby cluster) configuration:

```
jdbc:gaussdb://node1:port,node2:port,node3:port/database?isasterToleranceCluster=true&autoBalance=true
```

- **disasterToleranceCluster**: enables the system to automatically determine whether the target cluster is the primary cluster or standby cluster. The configured IP address must point to the IP address of the CN in the target cluster.
- **autoBalance**: enables load balancing.
  - If this parameter is set to **true**, **balance**, or **roundrobin**, the JDBC load balancing function is enabled to allocate multiple connections of an application to each CN available in the database cluster.
  - If this parameter is set to **priorityn**, the priority-based JDBC load balancing function is enabled to evenly allocate multiple connections of an application to the first *n* available CNs in the URL.
  - If this parameter is set to **shuffle**, JDBC random load balancing is enabled to randomly and evenly allocate multiple connections of the application to available CNs in the database cluster.
  - If this parameter is set to **specified**, the JDBC load balancing function is enabled and connections are established only on nodes specified in the user URL.
  - If this parameter is set to **shufflen**, JDBC random load balancing is enabled on specified nodes and multiple connections of an application are preferentially and randomly allocated to the first *n* available CNs in the URL. If all the first *n* nodes configured in the URL are unavailable, connections are randomly allocated to other available CNs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - If this parameter is set to **false**, the JDBC load balancing and priority-based load balancing functions are disabled.

NOTE

When distributed read on standby is enabled for the standby cluster in DR, **enableStandbyRead** cannot be set to **on**; otherwise, requests will be intercepted.

----End

7.27.3.3 Precautions

- Read on standby is designed to share the pressure of the primary node and reduce costs while ensuring that the replay RTO is normal. During the design and implementation, the replay on the standby node is preferentially ensured. The read success rate on the standby node can be increased by optimizing the configuration or using other methods. However, in actual service scenarios, it is impossible to ensure that the read on standby operation is 100% successful. Therefore, you are advised to deploy services that have high read requirements on the primary node.
- The conflict between long queries and prolonged RTO is inevitable on the standby node. Enabling ultimate RTO will increase the storage space and prolong the query holding without affecting the replay rate. Essentially, this is a space-time tradeoff policy. Long queries shall be avoided when you use the read on standby function.
- Read/write isolation can only be controlled at the session level. If a write operation is performed in a read on standby session (that is, **enable_standby_read** is set to **on**), the system throws an error. Note that this function does not support service switchover to the primary node when the standby node is abnormal.

7.27.4 FAQ

Queries Are Canceled in Case of a Query and Replay Conflict

- **Symptom**: When a query and replay conflict is detected, the replayer thread waits for a period of time specified by **max_standby_streaming_delay**. After the timeout interval expires, the replayer thread sends a cancellation instruction to the thread in conflict. As a result, the query in conflict is terminated and an error message is returned.

  The error details are as follows:

  User was holding shared buffer pin for too long.
  Triggering scenario: When VACUUM logs are replayed on the standby database, if pages in the buffer that the replayer thread needs to retrieve are queried by an active session that holds a pin lock, a buffer pin conflict occurs. The system cancels the query in conflict after waiting for a period of time specified by **max_standby_streaming_delay**. (Generally, this conflict rarely occurs because the query in conflict has been completed in advance.)

  User was holding a relation lock for too long.
  Triggering scenario: Operations such as DROP TABLE, TRUNCATE, and ALTER TABLE (in most cases) have a lock conflict with SELECT. If they are replayed on a standby database that is queried frequently, this lock conflict occurs. The waiting time is controlled by the **max_standby_streaming_delay** parameter.

  User was or might have been using tablespace that must be dropped.
  Triggering scenario: When a tablespace is dropped on the primary database while being queried on the standby database, for example, writing a temporary file to the tablespace, a conflict occurs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

User query might have needed to see row versions that must be removed.
Triggering scenario: The **max_standby_streaming_delay** parameter determines the policy for processing query and replay conflicts. In case of such a conflict, the standby node suspends log replay until the interval specified by the parameter expires, terminates the query in conflict if it is still running, and reports an error.

User transaction caused buffer deadlock with recovery.
Triggering scenario: When a conflict occurs between the replayer thread and the reader thread on the standby node, the system performs deadlock detection every 5 seconds before the conflict time reaches the **max_standby_streaming_delay** threshold.

User was connected to a database that must be dropped.
Triggering scenario: When **DROP DATABASE** is executed on the primary database, if an active session on the standby database connects to the database to be dropped, a conflict occurs.

- **Cause analysis**: In the query and replay conflict scenario, the system preferentially ensures that the replay function is normal and cancels the query in conflict. Replay operations that may cause such a conflict include VACUUM, DDL/DCL statements that hold a level-8 lock, DROP DATABASE, and DROP TABLESPACE.

  NOTE

  The ultimate RTO solution has removed such conflict on VACUUM, DROP TABLE, TRUNCATE TABLE (table only), and DROP DATABASE.

- In the ultimate RTO replay solution, when the primary node executes DDL statements and the standby node replays the corresponding DDL statements, the following blocking cases may occur.

Table 7-40

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User/ Table/ Database)	SQL Statement
add/ drop/ split/ exchange partition	add partition	Not blocked	ALTER TABLE test_p1 ADD PARTITION p4 VALUES LESS THAN (50);	Not blocked	ALTER TABLE test_p2 ADD PARTITION p4 VALUES LESS THAN (50);
	drop partition	Blocked	ALTER TABLE test_p1 DROP PARTITION p3;	Not blocked	ALTER TABLE test_p2 DROP PARTITION p3;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User / Table/ Database)	SQL Statement
	split partition	Blocked	ALTER TABLE test_p1 SPLIT PARTITION p4 INTO (PARTITION p4_1 VALUES LESS THAN (40), PARTITION p4_2 VALUES LESS THAN (50));	Not blocked	ALTER TABLE test_p2 SPLIT PARTITION p4 INTO (PARTITION p4_1 VALUES LESS THAN (40), PARTITION p4_2 VALUES LESS THAN (50));
	exchange partition	Blocked	CREATE TABLE test_ep1(col1 INT,col2 INT);	Not blocked	CREATE TABLE test_ep2(col1 INT,col2 INT);
			INSERT INTO test_ep1 VALUES (GENERATE_SERIES(1,5), 1000);		INSERT INTO test_ep2 VALUES (GENERATE_SERIES(1,5), 1000);
			ALTER TABLE test_p1 EXCHANGE PARTITION (p1) WITH TABLE test_ep1 VERBOSE;		ALTER TABLE test_p2 EXCHANGE PARTITION (p1) WITH TABLE test_ep2 VERBOSE;
create/ alter/ drop role/ user	creat role	Not blocked	CREATE ROLE tpcc1 WITH LOGIN PASSWORD '********';	Not blocked	CREATE ROLE tpcc2 WITH LOGIN PASSWORD '********';
	alter role	Not blocked	ALTER ROLE tpcc1 PASSWORD '********';	Not blocked	ALTER ROLE tpcc2 PASSWORD '********';
	drop role	Blocked	DROP ROLE tpcc1;	Not blocked	DROP ROLE tpcc2;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User / Table/ Database)	SQL Statement
			CREATE ROLE admin WITH LOGIN PASSWORD '********';		
			ALTER SCHEMA tpcc1 OWNER TO admin;		
			ALTER TABLE t1 OWNER TO admin;		
			REVOKE ALL PRIVILEGES ON SCHEMA public FROM tpcc1;		
			DROP ROLE tpcc1;		
	creat user	Not blocked	create user tpcc identified by '********';	Not blocked	create user tpcc3 identified by '********';
	alter user	Not blocked	ALTER USER tpcc IDENTIFIED BY ''********'';	Not blocked	ALTER USER tpcc3 IDENTIFIED BY ''********'';
	drop user	Blocked	drop user tpcc CASCADE;	Not blocked	drop user tpcc3 CASCADE;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User/ Table/ Database)	SQL Statement
create/ alter/ drop schema/ tablespace/ table/ index	create schema	Not blocked	CREATE SCHEMA school;	Not blocked	CREATE SCHEMA school1;
	alter schema	Not blocked	ALTER SCHEMA school RENAME TO middle_school;	Not blocked	ALTER SCHEMA school1 RENAME TO middle_school1;
	drop schema	Blocked	DROP SCHEMA middle_school CASCADE;	Not blocked	DROP SCHEMA middle_school1;
	create tablespace	Not blocked	create tablespace tablespace_school LOCATION '/home/perfadm';	Not blocked	create tablespace tablespace_school3 LOCATION '/var/standard/var/lib';
	alter tablespace	Blocked	create tablespace tablespace_school2 LOCATION '/var/standard/var/lib';	Not blocked	CREATE TABLE students3 (id INT, name VARCHAR2(100), age INT) TABLESPACE tablespace_school3;
			ALTER TABLE students SET TABLESPACE tablespace_school2;		create tablespace tablespace_school2 LOCATION '/var/standard/usr/local/core';
			-		ALTER TABLE students3 SET TABLESPACE tablespace_school2;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User / Table/ Database)	SQL Statement
	drop tablespace	Not blocked	DROP TABLESPACE tablespace_school2;	Blocked	DROP TABLESPACE tablespace_school2;
	create table	Not blocked	create table t1 (id int);	Not blocked	create table t2 (id int);
	alter table	Blocked	ALTER TABLE t1 ADD COLUMN age INT;	Not blocked	ALTER TABLE t2 ADD COLUMN age INT;
	drop table	Blocked	drop table t1;	Not blocked	drop table t2;
	create index	Not blocked	CREATE INDEX t1_index1 ON t1(id);	Not blocked	CREATE INDEX t2_index1 ON t2(id);
	alter index	Blocked	ALTER INDEX t1_index1 SET (fillfactor = 80);	Not blocked	ALTER INDEX t2_index1 SET (fillfactor = 80);
	drop index	Blocked	drop INDEX t1_index1;	Not blocked	drop INDEX t2_index1;
create/ alter/ drop sequence	create sequence	Not blocked	CREATE SEQUENCE seq1 START 101 INCREMENT 10;	Not blocked	CREATE SEQUENCE seq3 START 101 INCREMENT 10;
	alter sequence	Not blocked	ALTER SEQUENCE seq1 RESTART WITH 200;	Not blocked	ALTER SEQUENCE seq3 RESTART WITH 200;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User / Table/ Database)	SQL Statement
			ALTER SEQUENCE seq1 INCREMENT BY 2;		ALTER SEQUENCE seq3 INCREMENT BY 2;
	drop sequence	Blocked	DROP SEQUENCE seq1;	Not blocked	DROP SEQUENCE seq3;
grant/ revoke/ alter default privileges/ reassign owned	grant	Not blocked	GRANT SELECT ON TABLE t1 TO tpcc;	Not blocked	GRANT SELECT ON TABLE t1 TO tpcc;
			GRANT SELECT, INSERT ON t1 TO tpcc;		GRANT SELECT, INSERT ON t1 TO tpcc;
	revoke	Not blocked	REVOKE SELECT ON TABLE t1 FROM tpcc;	Not blocked	REVOKE SELECT ON TABLE t1 FROM tpcc;
	alter default privileges	Not blocked	ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO tpcc;	Not blocked	ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO tpcc;
	reassign owned	Blocked	CREATE ROLE admin WITH LOGIN PASSWORD '********';	Not blocked	CREATE ROLE admin WITH LOGIN PASSWORD '********';
			REASSIGN OWNED BY tpcc TO admin;		REASSIGN OWNED BY tpcc TO admin;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User/ Table/ Database)	SQL Statement
truncate partition	truncate partition	Blocked	ALTER TABLE test_p1 TRUNCATE PARTITION p1;	Not blocked	CREATE TABLE test_p2 (col1 INT, col2 INT) PARTITION BY RANGE (col1) (PARTITION p1 VALUES LESS THAN (10), PARTITION p2 VALUES LESS THAN (20), PARTITION p3 VALUES LESS THAN (30));
					INSERT INTO test_p2 (col1, col2) VALUES (5, 10);
					INSERT INTO test_p2 (col1, col2) VALUES (12, 20);
					INSERT INTO test_p2 (col1, col2) VALUES (20, 30);
					ALTER TABLE test_p2 TRUNCATE PARTITION p1;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation on the Primary Node		Verification Result (Current User/ Table/ Database)	SQL Statement	Verification Result (Non-Current User / Table/ Database)	SQL Statement
reindex	reindex	Blocked	REINDEX TABLE t1;	Not blocked	REINDEX TABLE t2;

- Solution: Retry the affected query.

A Deadlock Occurs Due to a Level-8 Lock Conflict in Read on Standby Scenarios

- **Symptom**: When multiple level-8 lock logs are generated on the primary node in a short period of time, the read on standby operation may trigger a session thread deadlock and throw the error "GAUSS-23827: deadlock detected."
- **Cause analysis**: When the replayer thread on the standby node continuously processes level-8 lock logs, it holds the level-8 lock of relation 1 and attempts to obtain the level-8 lock of relation 2; however, the query thread holds the level-1 lock of relation 2 and attempts to obtain the level-1 lock of relation 1, forming a deadlock due to mutual waiting. Statements that generate level-8 lock logs include ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL, and LOCK TABLE that does not declare other lock modes.
- **Solution**: Retry the operation.

A Lock Conflict Occurs Between Standby Index Scan and Log Replay

- **Symptom**: In serial/parallel replay scenarios, the following error is reported when the standby node executes read services: "Index scan conflicts with replay. Try again later. The original maxoff is, but now it is."
- **Cause analysis**: When a query thread performs index scan on the standby node, it first adds a read lock to index pages and checks the visibility of tuples one by one during the scan. If the transactions corresponding to the tuples are being committed, the thread waits until the transactions are committed. Transaction commit on the standby node replies on log replay and the log replayer thread modifies index pages in this process, which requires a lock. In this case, the query thread must proactively release the obtained index page lock during the waiting process. Otherwise, a deadlock occurs in which the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

query thread waits for the replayer thread to commit the transaction and the replayer thread waits for the query thread to release the lock.

- **Workaround**: Retry on the service side.

Query on the Standby Node Is Abnormal Due to Asynchronous Metadata on the Primary and Standby Nodes

- **Symptom**: During read on standby in serial/parallel replay scenarios, the error message "relation \"<string>\" does not exist" or "column \"<string>\" does not exist" is displayed.
- **Cause analysis**: DDL operations modify metadata, causing metadata inconsistency between the primary and standby nodes. In this case, the read operation on the standby node will trigger the preceding error or other error messages due to metadata inconsistency. In this case, wait until the standby node replays the metadata logs and synchronizes with the primary node in terms of the metadata status. Then, the read on standby operation can be performed properly.
- **Workaround**: Verify that the metadata on the primary and standby nodes is synchronized before performing the read on standby operation.

7.28 Anti-overloads

7.28.1 SQL Statement Concurrency Control

7.28.1.1 Overview

Scenarios

When the database system is running, service request traffic may burst too high or some SQL statements may occupy too many resources. SQL statement concurrency control can limit the execution of such SQL statements to prevent them from exhausting database resources and affecting the execution of normal service SQL statements.

Description

This feature can limit the concurrency of a certain type of SQL statements at the CN process level or all SQL statements based on multiple rules, including:

- Concurrency control based on the unique SQL ID: If an SQL statement is slow or occupies a large number of resources, you can use the unique SQL ID to control the SQL statement concurrency to prevent other services from being affected by a large number of SQL statements.
- Concurrency control based on SQL types and keywords: When a certain type of SQL requests may increase with the service volume, you can use SQL types and keywords to control the SQL statement concurrency.
- Concurrency control based on client information: If the services of a certain client occupy a large number of resources, you can use the client information, including the IP address and application name, to control the concurrency,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

preventing other services from being affected by the execution of a large number of SQL statements on the client.

- Some service peaks are predictable. If you want to control SQL requests only during peak hours, you can set the effective time of SQL statement concurrency control to prevent concurrency control rules from being resident in the system.
- If you want the concurrency control rules to apply only to the service database and do not control the SQL statements of the system database, you can control the concurrency by database dimension.
- In addition to the SQL statement concurrency of a certain type, this feature also controls the CN process-level concurrency.
- It provides the capability of querying and collecting statistics on concurrency control rules. You can query all concurrency control rules and manage concurrency control rules based on the rule list. In addition, you can query the number of times that the concurrency control rules take effect.

Specifications and Limitations

- The concurrency control is performed only on CNs.
- For concurrency control based on Unique SQL ID, set **enable_resource_track** to **on** and **instr_unique_sql_count** to a value greater than 0.
- The concurrency control based on keywords is sorted by concurrency. The lower the concurrency, the higher the priority. Keywords are case-insensitive and support fuzzy match.
- For concurrency control based on client information, exact match rules are preferentially used (both the IP address and application name must be matched). If there is no effective exact match rule, the rules that match only IP addresses or application names are sorted by concurrency in ascending order. A lower concurrency indicates a higher priority. All keywords are case-sensitive and do not support fuzzy match.
- The database name is case-sensitive. If a database is deleted and a database with the same name is created, all recorded concurrency control rules for the database become invalid.
- The concurrency control based on the maximum number of active concurrent statements at the CN process level of resources is sorted by concurrency. The lower the concurrency, the higher the priority. There is a time difference of about 10s for collecting resource usage.
- For a rule whose concurrency control period ends, the system marks the **is_valid** flag in the cache as **false** when the next concurrency control rule is triggered. The **is_valid** column in the system catalog is detected and updated by a background thread, which may be delayed for 10 minutes at maximum.
- If a flow control rule is added, deleted, or modified in a transaction and the transaction is finally rolled back, the data in the flow control rule cache will be inconsistent with that in the system catalog of the flow control rule. In this case, you need to call the gs_refresh_workload_rule_cache function to update the flow control rule cache promptly.
- The number of SQL statement concurrency control times is the accumulated number of times from the startup of the database to the current time. After the database is restarted, the number of times is counted from 0.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Concurrent SQL statements in cursors and stored procedures are not controlled.
- The concurrent SQL statements performed by the administrator are not controlled.
- Concurrency control rules are not shared among databases. When creating a concurrency control rule, you need to connect to the target database. If the CN where a concurrency control rule is created is removed and a full build is triggered, the concurrency control rule in the target CN of the full build is inherited. Therefore, you are advised to create a concurrency control rule on each CN.
- CNs do not share concurrency control rules because their unique SQL IDs are different. Users need to manually create concurrency control rules on different CNs.

7.28.1.2 Technical Principle

Core Mechanism

When GaussDB is started, SQL statement concurrency control rules are loaded from the system catalog to the memory. The process of SQL statement concurrency control is as follows:

1. When a user executes any SQL statement, GaussDB matches the SQL statement with the concurrency control rules in the memory.
2. If a concurrency control rule is matched, the value of the **dop** field of the rule in the memory is increased by 1. The **dop** field records the real-time concurrency of the concurrency control rule.
3. If the value of the **dop** field of the matched concurrency control rule is greater than or equal to the maximum concurrency allowed by the rule, the concurrency control is performed: The SQL statement execution is terminated immediately, and an error is reported to the user.
4. When the concurrency control takes effect or the SQL statement is executed successfully without concurrency control taking effect, the value of the **dop** field of the matched concurrency control rule is decreased by 1.

API Description

SQL statement concurrency control uses the following system functions. For details, see "SQL Reference > Functions and Operators > SQL Statement Concurrency Control Functions" in *Reference*.

Function	Description
gs_add_workload_rule	Creates an SQL statement concurrency control rule. Users must have the SYSADMIN permission.
gs_update_workload_rule	To update an SQL statement concurrency control rule, users need to reset all parameters instead of only some parameters. Users must have the SYSADMIN permission.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Description
gs_delete_workload_rule	Deletes an SQL statement concurrency control rule. Users must have the SYSADMIN permission.
gs_get_workload_rule_stat	Queries the number of times that SQL statements are intercepted by SQL statement concurrency control rules. Users must have the SYSADMIN permission.
gs_refresh_workload_rule_cache	Refreshes the concurrency control rule cache to ensure that the cache is consistent with the system catalog of the concurrency control rule. It cannot be executed in a transaction. Users must have the SYSADMIN permission.

7.28.1.3 Operation Guide

Prerequisites

- GaussDB version ≥ 505.0.0
- Only users with the SYSADMIN permission can perform the following operations.

Examples

```
-- Create a rule named rule_1 to limit the maximum concurrency of SQL statements whose unique SQL ID
is 844424930505717 to 10 immediately.
gaussdb=# SELECT gs_add_workload_rule('sqlid', 'rule_1', '{}', NULL, NULL, 10, '{id=844424930505717}');
 gs_add_workload_rule
----------------------
                    1
(1 row)

-- Create a rule to limit the maximum concurrency of SQL statements of the select type in the postgres
and tpcc databases with the keywords group by and join to 1 during the specified period.
gaussdb=# SELECT gs_add_workload_rule('select', 'rule_2', '{postgres, tpcc}', '2026-01-01 00:00:00',
'2026-01-02 00:00:00', '1', '{group by, join}');
 gs_add_workload_rule
----------------------
                    2
(1 row)

-- Create a rule to limit the maximum concurrency of SQL statements to 10 when the CPU usage is greater
than 60% and the memory usage is greater than 80%.
gaussdb=# SELECT gs_add_workload_rule('resource', 'rule_3', '{}', NULL, NULL, '10', '{cpu-60, memory-80}');
 gs_add_workload_rule
----------------------
                    3
(1 row)

-- Create a rule named rule_4 to limit the maximum concurrency of SQL statements to 20 based on the
client application app1. The rule takes effect immediately.
gaussdb=# SELECT gs_add_workload_rule('client', 'rule_4', '{}', NULL, NULL, 20, '{appname=app1}');
 gs_add_workload_rule
----------------------
                    4
(1 row)

-- Modify the SQL statement concurrency control rule whose rule_id is 1 (the rule is based on the unique
```

```
SQL ID). Change the unique SQL ID to 676844332506578, maximum concurrency to 20, and effective time
to a specified period.
gaussdb=# SELECT gs_update_workload_rule(1, 'rule_1', '{}', '2026-01-01 00:00:00', '2026-01-02 00:00:00',
20, '{id=676844332506578}');
 gs_update_workload_rule
-------------------------
 t
(1 row)

-- Refresh the concurrency control rule cache.
gaussdb=# SELECT gs_refresh_workload_rule_cache();
 gs_refresh_workload_rule_cache
--------------------------------
 t
(1 row)

-- Query the statistics of the SQL statement concurrency control rule whose rule_id is 1.
gaussdb=# SELECT * from gs_get_workload_rule_stat(1);
 rule_id | validate_count
---------+----------------
       1 |              0
(1 row)

-- Query the statistics of all SQL statement concurrency control rules.
gaussdb=# SELECT * from gs_get_workload_rule_stat(-1);
 rule_id | validate_count
---------+----------------
       1 |              0
       2 |              0
       3 |              0
       4 |              0
(4 rows)

-- Delete the SQL statement concurrency control rule whose rule_id is 1.
gaussdb=# SELECT gs_delete_workload_rule(1);
 gs_delete_workload_rule
-------------------------
 t
(1 row)

-- Delete all SQL statement concurrency control rules.
gaussdb=# SELECT gs_delete_workload_rule(rule_id) FROM gs_workload_rule;
 gs_delete_workload_rule
-------------------------
 t
 t
 t
(3 rows)
```

7.28.1.4 Typical Issues

Issue: The gs_workload_rule system catalog cannot be found.

Solution:

The SQL statement concurrency control feature is introduced in 505.0.0. Ensure that the database version is 505.0.0 or later.

Issue: SQL statement concurrency control rules are set, but SQL statement concurrency control is not triggered when SQL statements are executed.

Solution:

Perform the following steps:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Check whether the value of the GUC parameter **enable_workload_rule** is **on**. If the value of this parameter is **off**, SQL statement concurrency control does not take effect.
- Check whether the user who executes the SQL statement is an administrator. SQL statement concurrency control does not take effect for administrators.
- Check whether the user who executes the SQL statement is in the SQL statement concurrency control whitelist. Whitelisted users will not trigger SQL statement concurrency control. The check method is as follows:
  ```
  gaussdb=# SHOW workload_uncontrolled_users;
  ```
- Check whether the SQL statement concurrency control rule is correctly set. For example, check whether the SQL ID, database, and keyword are correct, whether the SQL statement concurrency control rule has expired, whether the start time of SQL statement concurrency control is earlier than the end time, and whether **max_workload** is correctly set.

7.28.2 Memory Overload Protection

7.28.2.1 Overview

Description

When the database memory is overloaded, sessions are closed instantly and new connections are forbidden till the memory is recovered to provide services.

Scenarios

When the SQL statement execution latency increases significantly due to an IaaS layer exception, a large number of sessions are stacked, causing memory resource exhaustion and preventing the database from providing services. In this case, the feature enables the database to quickly escape and recover its services within a short period of time.

NOTE

The Infrastructure as a Service (IaaS) layer provides basic computing, storage, and network services. Typical IaaS cloud services include Elastic Cloud Server (ECS).

Specifications and Limitations

- When memory overload triggers escape, sessions of users with the SYSADMIN or MONADMIN permission are not cleared by default. To clear these sessions, set the GUC parameter **resilience_escape_user_permissions**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.
- In upgrade mode, this feature is not triggered.

7.28.2.2 Technical Principle

Core Mechanism

The GUC parameter **resilience_memory_reject_percent** sets two thresholds: **recover_memory_percent** and **overload_memory_percent**. For details, see

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

"Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

The wlm_monitor thread periodically checks whether the dynamic memory usage is greater than the value of **overload_memory_percent**. If so, escape from overload is triggered to prevent subsequent session access. Idle sessions are preferentially closed. For the sessions in the same status, those with the highest memory usage are closed first. When the dynamic memory usage falls below the value of **recover_memory_percent**, the system stops closing sessions and allows connections.

Dynamic memory usage = **dynamic_used_memory**/**max_dynamic_memory** × 100%. **dynamic_used_memory** and **max_dynamic_memory** can be obtained by querying the PV_TOTAL_MEMORY_DETAIL view. For details, see "System Catalogs and System Views > System Views > Other System Views > PV_TOTAL_MEMORY_DETAIL" in *Reference*.

You can query the PG_STAT_ACTIVITY view to obtain the session status. For details, see "System Catalogs and System Views > System Views > Other System Views > PG_STAT_ACTIVITY" in *Reference*.

API Description

The following GUC parameter is used for memory overload protection. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

Table 7-41 GUC parameter

Parameter	Description
use_workload_manager	Specifies whether to enable the resource management functionality. This parameter must be enabled for memory overload protection to take effect.

The following GUC parameters are used for memory overload protection. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

Table 7-42 GUC parameters

Parameter	Description
enable_memory_limit	Specifies whether to enable the memory management functionality. This parameter must be enabled for memory overload protection to take effect.
resilience_memory_reject_percent	Specifies the dynamic memory usage percentage for escape from memory overload.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
resilience_escape_user_permissions	Specifies whether to enable the memory overload protection functionality for users with the SYSADMIN or MONADMIN permission.

7.28.2.3 Operation Guide

Prerequisites

The GUC parameters **use_workload_manager** and **enable_memory_limit** are enabled (default setting). You need to manually set **resilience_memory_reject_percent**.

Procedure

Step 1 Check whether the **enable_memory_limit** parameter is enabled.

```
gaussdb=# SHOW enable_memory_limit;
 enable_memory_limit
---------------------
 on
(1 row)
```

Step 2 Check whether the **use_workload_manager** parameter is enabled.

```
gaussdb=# SHOW use_workload_manager;
 use_workload_manager
----------------------
 on
(1 row)
```

Step 3 Set the **resilience_memory_reject_percent** parameter. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

Step 4 To set overload escape for users with the SYSADMIN or MONADMIN permission, set the **resilience_escape_user_permissions** parameter. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

----End

7.28.2.4 Typical Issues

Issue: Escape from overload does not take effect. As a result, the dynamic memory usage of the database is higher than the escape threshold.

Solution: Check whether **use_workload_manager** and **enable_memory_limit** are enabled. If they are disabled, the functionality does not take effect.

```
gaussdb=# SHOW enable_memory_limit;
 enable_memory_limit
---------------------
 on
(1 row)

gaussdb=# SHOW use_workload_manager;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 use_workload_manager
----------------------
 on
(1 row)
```

Check whether the **resilience_memory_reject_percent** parameter is successfully set. If the recovery threshold is greater than the threshold for triggering escape, the parameter setting fails and the default value is used. Check whether the service running user has the SYSADMIN or MONADMIN permission in the system view pg_roles. By default, escape from overload will not be triggered for users with the SYSADMIN or MONADMIN permission.

```
gaussdb=# SHOW resilience_memory_reject_percent;
 resilience_memory_reject_percent
----------------------------------
 50,90
(1 row)

gaussdb=# SELECT rolname, rolsystemadmin, rolmonitoradmin FROM pg_roles WHERE rolname = 'Mike';
            rolname             | rolsystemadmin | rolmonitoradmin
--------------------------------+----------------+-----------------
 Mike                           | t              | t
(1 rows)

gaussdb=# SHOW resilience_escape_user_permissions;
 resilience_escape_user_permissions
------------------------------------

(1 row)
```

Issue: Escape from overload is frequently triggered.

Solution: Check whether the value of **resilience_memory_reject_percent** is too small. If it is too small, raise the overload threshold.

7.28.3 Thread Pool Overload Protection

7.28.3.1 Overview

Description

When the database thread pool is overloaded, sessions are quickly closed and new connections are forbidden till the thread pool is recovered to provide services.

Scenarios

When the SQL statement execution latency increases significantly due to an IaaS layer exception, a large number of sessions are stacked, causing thread pool resource exhaustion and preventing the database from providing services. In this case, the feature enables the database to quickly escape and recover its services within a short period of time.

NOTE

The Infrastructure as a Service (IaaS) layer provides basic computing, storage, and network services. Typical IaaS cloud services include Elastic Cloud Server (ECS).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifications and Limitations

- This feature takes effect when the GUC parameters **use_workload_manager** and **enable_thread_pool** are enabled.
- When thread pool overload triggers escape, sessions of users with the SYSADMIN or MONADMIN permission are not cleared by default. To clear these sessions, set the GUC parameter **resilience_escape_user_permissions**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.
- This feature is not triggered during the upgrade observation period.

7.28.3.2 Technical Principle

Core Mechanism

The GUC parameter **resilience_threadpool_reject_cond** sets two thresholds: **recover_threadpool_percent** and **overload_threadpool_percent**.

The wlm_monitor thread periodically checks whether the thread pool usage is greater than the value of **overload_threadpool_percent**. If so, escape from overload is triggered to prevent subsequent session access. Idle sessions are preferentially closed. For the sessions in the same status, those with earliest query start time are closed first. When the thread pool usage drops below the value of **recover_threadpool_percent**, the system stops closing sessions and allows new connections.

You can query the DBE_PERF.LOCAL_THREADPOOL_STATUS view to obtain the thread pool usage status. For details, see "Schema > DBE_PERF Schema > Session/Thread > DBE_PERF.LOCAL_THREADPOOL_STATUS" in *Reference*.

You can query the PG_STAT_ACTIVITY view to obtain the session status and the start time of the query. For details, see "System Catalogs and System Views > System Views > Other System Views > PG_STAT_ACTIVITY" in *Reference*.

API Description

The following GUC parameters are used for thread pool overload protection. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

Table 7-43 GUC parameter

Parameter	Description
use_workload_manager	Specifies whether to enable the resource management functionality. This parameter must be enabled for thread pool overload protection to take effect.

The following GUC parameters are used for thread pool overload protection. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-44 GUC parameter

Parameter	Description
resilience_escape_user_permissions	Specifies whether to enable the thread pool overload protection functionality for users with the SYSADMIN or MONADMIN permission.

The following GUC parameters are used for thread pool overload protection. For details, see "Database GUC Parameters > GUC Parameters > Thread Pool" in *Reference*.

Table 7-45 GUC parameters

Parameter	Description
enable_thread_pool	Specifies whether to enable the thread pool functionality.
resilience_threadpool_reject_cond	Specifies the percentage of thread pool usage for escape from overload.

7.28.3.3 Operation Guide

Prerequisites

The GUC parameters **use_workload_manager** and **enable_thread_pool** are enabled (default setting). **resilience_threadpool_reject_cond** needs to be manually set.

Procedure

Step 1 Check whether the **enable_thread_pool** parameter is enabled.

```
gaussdb=# SHOW enable_thread_pool;
 enable_thread_pool
--------------------
 on
(1 row)
```

Step 2 Check whether the **use_workload_manager** parameter is enabled.

```
gaussdb=# SHOW use_workload_manager;
 use_workload_manager
----------------------
 on
(1 row)
```

Step 3 Set the **resilience_threadpool_reject_cond** parameter. For details, see "Database GUC Parameters > GUC Parameters > Thread Pool" in *Reference*.

Step 4 To set overload escape for users with the SYSADMIN or MONADMIN permission, set the **resilience_escape_user_permissions** parameter. For details, see "Database

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

----End

7.28.3.4 Typical Issues

Issue: Escape from overload is frequently triggered.

Solution: Check whether the value of **resilience_threadpool_reject_cond** is too small. If it is too small, raise the overload threshold.

Issue: Escape from overload does not take effect. As a result, the thread pool usage is higher than the escape threshold.

Solution: Check whether the **use_workload_manager** and **enable_thread_pool** parameters are enabled. If not, the escape from overload functionality does not take effect. If they are enabled, check whether the **resilience_threadpool_reject_cond** parameter is successfully set.

```
gaussdb=# SHOW enable_thread_pool;
 enable_thread_pool
--------------------
 on
(1 row)

gaussdb=# SHOW use_workload_manager;
 use_workload_manager
----------------------
 on
(1 row)

gaussdb=# SHOW resilience_threadpool_reject_cond;
 resilience_threadpool_reject_cond
-----------------------------------
 50,90
(1 row)
```

Issue: After escape from overload is triggered, the connection cannot be restored.

Solution: Check whether users with the SYSADMIN or MONADMIN permission occupy too many threads. By default, using this functionality will not close the sessions of users with the SYSADMIN or MONADMIN permissions. After escape from overload is triggered, the connection may not be restored. You can set **resilience_escape_user_permissions** to control the escape of users with the SYSADMIN or MONADMIN permission.

7.28.4 Connection Thread Pool

7.28.4.1 Overview

Scenarios

In many cases, when the database system is running, a large number of connections are established concurrently, preempting the thread pool resources. As a result, the service throughput decreases greatly due to lack of thread pool resources. Adding a connection thread pool can reduce the impact of connection

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

access on the SQL execution performance of established services, preventing slow service execution or sharp throughput drop.

Description

An independent thread pool is split from the existing thread pool to process database connection establishment. It decouples services and connection establishment, preventing resource contention in the same thread pool.

Specifications and Limitations

- Connections accessed through the O&M port do not use the connection thread pool.
- The connection thread pool functionality can be enabled only when both GUC parameters **enable_connect_thread_pool** and **enable_thread_pool** are set to **on**.
- In upgrade mode, this feature is not triggered.

7.28.4.2 Technical Principle

Working Principle

After the connection thread pool functionality is enabled, the connection establishment process changes. When GaussDB receives a connection establishment event, it dispatches the task to the connection thread pool. The worker thread in the thread pool binds the session corresponding to the connection to execute initialization. After the initialization is complete, the SQL statement in the connection establishment phase is executed. After the connection is established, the session is dispatched to the worker thread pool to execute subsequent services. The process is as follows:

1. The client (such as gsql or JDBC) triggers a connection establishment operation.
2. After receiving the connection establishment request, GaussDB creates a session and dispatches it to the connection thread pool.
3. The connection thread pool obtains an idle ConnTPLWorker thread, binds it to the session, and performs connection initialization, including authentication and first packet processing.
4. After the connection is initialized, the ConnTPLWorker thread continues to execute the SQL statements in the connection establishment phase.
5. After the connection is established, the ConnTPLWorker thread dispatches the session to the worker thread pool for the next task.
6. The worker thread pool obtains an idle TPLWorker thread, binds it to the session, executes subsequent SQL services, and returns the execution result.

Key APIs

The following GUC parameters are used for the connection thread pool. For details, see "Database GUC Parameters > GUC Parameters > Thread Pools" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-46 GUC parameters

Parameter	Description
enable_thread_pool	Specifies whether to enable the thread pool. The default value is **on**.
enable_connect_thread_pool	Specifies whether to enable the connection thread pool. The default value is **on**.
connect_thread_pool_attr	Initial configuration parameter of the connection thread pool. The default value is **'auto'**.

7.28.4.3 Operation Guide

Prerequisites

The GUC parameters **enable_connect_thread_pool** and **enable_thread_pool** are set to **on**.

Procedure

Step 1 Check whether the **enable_connect_thread_pool** parameter is enabled.

```
gaussdb=# show enable_connect_thread_pool;
 enable_connect_thread_pool
----------------------------
 on
(1 row)
```

Step 2 Check whether the **enable_thread_pool** parameter is enabled.

```
gaussdb=# show enable_thread_pool;
 enable_thread_pool
--------------------
 on
(1 row)
```

Step 3 Check the configuration of the connection thread pool parameter. If the value of **group_id** is **-1**, the thread pool is used for connection establishment.

```
gaussdb=# SELECT * from DBE_PERF.LOCAL_THREADPOOL_STATUS where group_id='-1';
 node_name  | group_id | bind_numa_id | bind_cpu_number | listener |
worker_info                          |              session_info
|         stream_info
------------+----------+--------------+-----------------+----------
+-------------------------------------------------------------+----------------------------------------
+-----------------------------
 singlenode |       -1 |              |               0 |        1 | default: 8 new: 0 expect: 8 actual: 8 idle: 8 pending: 0 |
total: 0 waiting: 0 running:0 idle: 0
| total: 0 running: 0 idle: 0
(1 row)
```

----End

7.28.5 Overload Escape in Concurrency Scenarios

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.28.5.1 Overview

Description

Identifies SQL statements whose execution time exceeds a specified time as slow SQL statements, and limits the percentage of slow SQL statements in the total threads of the thread pool and the IOPS of slow SQL statements.

Scenarios

There are too many slow SQL statements. As a result, no idle thread pool thread is available for normal SQL statements.

Specifications and Limitations

Functionality restrictions:

- The GUC parameter **enable_thread_pool** must be set to **on**.
- This parameter is valid only for SELECT statements executed by non-SYSADMIN/MONADMIN users.
- To limit the IOPS, the GUC parameter **use_workload_manager** must be set to **on**.

7.28.5.2 Technical Principle

Core Mechanism

Set the GUC parameter **resilience_ctrlstmt_detect_timelimit** or **wlmrule Plan Hint**. The worker thread sets a timer during the execution. After the timer expires, the current worker thread is identified as the thread for executing slow SQL statements. For the thread that executes slow SQL statements, the following management is performed:

- When the number of threads that execute slow SQL statements exceeds the value of **resilience_ctrlslot_available_maxpercent** multiplied by the maximum number of threads in the thread pool, subsequent threads that execute slow SQL statements are forcibly canceled.
- When the IOPS of the thread that executes slow SQL statements exceeds **resilience_ctrlstmt_control_iopslimit**, the corresponding thread sleeps for a random period ranging from 0s to 1s.

API Description

The following GUC parameter is used for fast/slow lanes. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-47 GUC parameters

Parameter	Description
resilience_ctrlslot_available_maxpercent	Specifies the maximum percentage of threads in the thread pool that can be occupied by slow SQL statements.
resilience_ctrlstmt_detect_timelimit	Specifies the execution time of a normal SQL statement after which it is marked as a slow SQL statement. The value **0** indicates that slow SQL statements are not identified. A value greater than 0 indicates that the SQL statement is marked as a slow SQL statement when its execution time exceeds the value of this parameter.
resilience_ctrlstmt_control_iopslimit	Specifies the maximum IOPS that can be used by slow SQL statements after normal SQL statements are marked as slow SQL statements.

The fast/slow lanes use the **wlmrule Plan Hint**. For details, see "Hint-based Optimization > Hints for Setting Slow SQL Control Rules" in *SQL Optimization Guide*.

Table 7-48 Plan Hint

Plan Hint	Plan Hint Description
wlmrule("time_limit, max_execute_time,max_iops")	Users can set the execution time, maximum execution time, and maximum IOPS for SQL statements marked as slow SQL statements.

7.28.5.3 Operation Guide

Prerequisites

The GUC parameter **enable_thread_pool** is set to **on**. To limit the IOPS of slow SQL statements, set the GUC parameter **use_workload_manager** to **on**.

Precautions

- The GUC parameters take effect globally, but **wlmrule Plan Hint** takes effect only for the SQL statements that are set.
- The settings of the GUC parameters **resilience_ctrlstmt_detect_timelimit** and **resilience_ctrlstmt_control_iopslimit** overlap with those of **wlmrule Plan Hint**. If both the corresponding GUC parameters and **wlmrule Plan Hint** are set for SQL statements, only the settings of **wlmrule Plan Hint** take effect. Otherwise, the corresponding GUC parameters take effect.
- You can also set the maximum execution time of SQL statements using **max_execute_time** in **wlmrule Plan Hint**. If the execution time of an SQL

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

statement exceeds the maximum execution time, the SQL statement is forcibly canceled.

- When the GUC parameter **resilience_ctrlstmt_detect_timelimit** is set to **0**, the functionalities of the **resilience_ctrlslot_available_maxpercent** and **resilience_ctrlstmt_control_iopslimit** parameters cannot take effect even if they have values.

Procedure

For details about how to set the GUC parameters, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

For details about how to set **wlmrule Plan Hint**, see "Hint-based Optimization > Hints for Setting Slow SQL Control Rules" in *SQL Optimization Guide*.

7.28.5.4 Typical Issues

Issue: Why does I/O control not take effect after **resilience_ctrlstmt_control_iopslimit** is set?

Solution: Check whether the value of the GUC parameter **use_workload_manager** is **on**. The thread for executing slow SQL statements sleeps for a random period ranging from 0 to 1 second. As a result, the IOPS cannot be completely controlled at a value below **resilience_ctrlstmt_control_iopslimit**.

7.29 Unix Domain Socket O&M Channel

7.29.1 Overview

Description

This emergency link provided by GaussDB allows the client to deliver emergency commands by accessing the link. This functionality is called the emergency channel for short.

Scenarios

- In extreme load scenarios, such as CPU overload, memory overload, full thread pool, and full connection pool, the cluster management component (CM) may fail to connect to the database through network connection (libpq) to check the database availability. As a result, a primary/standby failover is triggered. However, using the emergency channel can prevent the failover.
- O&M personnel may fail to connect to the database through the O&M port. As a result, some key O&M commands cannot be delivered promptly. In this case, the emergency channel can be used to deliver key commands.

Specifications and Limitations

- Only CN/DNs are supported. The log node in the networking where one primary node, one standby node, and one log node are deployed is not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- This requirement supports the following scenarios:
  - The peak CPU load of the GaussDB process reaches 95%.
  - The maximum number of connections reaches the upper limit of **max_connections**, and no new connection is allowed. For details about the GUC parameter **max_connections**, see "Database GUC Parameters > GUC Parameters > Connection and Authentication > Connection Settings" in *Reference*.
  - The memory is overloaded, and the message "memory is temporarily unavailable" is repeatedly reported in logs. As a result, no more memory can be allocated to execute SQL statements.
  - The thread pool is full, sessions are stacked, and new sessions are processed slowly.
- gsql is used to query emergency channel views or perform O&M. The result is returned within 5 seconds. Mem_log generation is performed asynchronously to ensure it is triggered within 5 seconds; however, the completion time depends on disk I/O, the number of memory contexts, and service concurrency. Similarly, session scanning and killing are executed asynchronously to ensure that signals are issued within 5 seconds. The success and duration of the process are affected by factors such as executed services and lock resources.
- Emergency channel O&M supports only local connection nodes.
- To record logs in **mem_log** using gsql, you need to enable **use_workload_manager** and **enable_memory_limit**. For details, see **Operation Guide**.
- In high CPU usage scenarios, ensure that the **enable_control_group** parameter is enabled (this is the default setting and CPU isolation can be performed) to ensure that O&M operations of the emergency channel are 100% successful. For details about the **enable_control_group** parameter, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

7.29.2 Technical Principles

Core Mechanism

- Communication between processes

  The interaction link between the kernel and CM is not subject to network or high load problems. The interaction is implemented through inter-process communication, which uses UDS.

  When analyzing the CM liveness detection scenario, consider the following points:
  - Indicator timeliness: CM liveness detection indicators are frequently collected, with some being collected every 1s. The shared memory leads to delay, and the timeliness of indicators needs to be determined. The UDS is highly real-time, which is more suitable for this scenario.
  - Variable parameters: The original CM liveness detection is implemented by executing SQL statements after network connection is established. The CM dynamically concatenates SQL statements through parameters. The kernel cannot predict input parameters in advance and collect them. In

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

this scenario, the CM needs to place the query command in the shared memory. After reading the command, the kernel executes the indicator query operation, which finally leads to a longer delay.

Considering the preceding factors, the UDS is used to implement the interaction between the CM and kernel.

- Resource reservation via the emergency channel

  The kernel reserves resources via the emergency channel, so that the liveness detection capability is still available in extreme load scenarios. In addition, this method minimizes I/O dependency during the collection/O&M of liveness detection indicators.

  - Reserving CPU resources: Bind the HA_BACKEND Cgroup.
  - Reserving memory resources: The emergency channel thread uses little memory, but has high priority. Therefore, the memory context usage is not limited. Before querying views or performing O&M operations through the emergency channel thread, switch the memory context.

- Main process of interaction between the kernel, emergency channel thread, and CM (gsql)
  - The CM and kernel initialize the UDS.
  - A persistent connection is established between the CM and kernel.
  - Threads in the CM periodically send packets to the kernel as required to query indicators or deliver O&M instructions after arbitration.
  - The kernel parses packets, deserializes parameters, and performs query or O&M operations based on instructions. For O&M operations, the command logic determines whether the synchronous or asynchronous API is used. For indicator query, the synchronous API is used. After obtaining the result set, the kernel serializes, encapsulates, and sends packets. In particular, when obtaining the information required by the CM, the kernel ensures that only the spinlock and a few necessary hash table locks are added.

  Considering that the persistent connection may be interrupted or blocked due to exceptions, the following mechanisms are required to ensure that the liveness detection link is normal:

  - The kernel liveness detection thread must have a keepalive mechanism. If the thread exits abnormally, it should be restarted promptly.
  - If the link between the CM and kernel is disconnected unexpectedly, the CM should attempt to establish the connection for a certain number of times. If these attempts fail, the node status is arbitrated based on the connection establishment and liveness detection results of the original network.
  - A timeout and retry mechanism should be set for the communication between the CM and kernel. If no result is returned after the request is retried for a certain number of times, the node status is arbitrated based on the connection establishment and liveness detection results of the original network.

- Liveness detection mechanism

  The original core process of the CM to detect liveness through network links is to send **SELECT 1** after connection establishment. When the liveness detection through network links fails in high load scenarios, the emergency

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

channel emerges as a substitute. Therefore, a new liveness detection instruction is added to the kernel. The implementation is as follows: The wlm_monitor thread asynchronously executes **SELECT 1** and records the result, covering link detection for all SQL execution processes except for the connection establishment. The emergency channel thread obtains the result and returns it to the CM.

- Simple O&M commands

  gsql provides simple O&M metacommands. When the gsql client executes a simple O&M command, the command is translated into a specified SQL statement and sent to the server.

API Description

- GUC parameters

  The UDS O&M channel uses the following GUC parameters. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

  Table 7-49 GUC parameters

Parameter	Description
enable_emergency_access	Specifies whether to enable the UDS O&M channel.

- O&M commands of the emergency channel

  The gsql tool provides the **-O** option to execute O&M commands of the emergency channel. For details, see **Operation Guide**. For details about how to use the gsql tool, see "Tool Reference > Database Connection Tools > gsql for Connecting to a Database > gsql Usage Guide" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-50 O&M commands of the UDS O&M channel supported by gsql

Parameter	Description	Value Range
-O, --emergency-command=EMERGENCYCOMMAND	O&M command of the UDS O&M channel. If gsql cannot connect to GaussDB in extreme cases, you can run this command for emergency O&M. This command communicates with GaussDB through the UDS O&M channel to implement one-click O&M. The parameters of the **-O** option are described as follows: "{port} {command} [{param} [...]]" *{port}*: database listening port. *{command}*: specific O&M command. *{param}*: parameter of the O&M command. There may be zero or more parameters. **NOTE** • When this option is used, gsql must be located on the same server as the GaussDB process. • If the parameter of the O&M command contains spaces, enclose the parameter with quotation marks.	• **"*{port}* mem_log"**: records the global memory overview, including details about the top 20 memory context usage at the global, thread, and session levels. This function is valid only when the GUC parameter **use_workload_manager** is enabled. The range is the same as that of the **mem_log** feature (recording memory overloading). There is no parameter. Example: -O "19995 mem_log" • **"*{port}* kill_wait_event *{wait_event}*"**: kills the session of a specified wait event. The *{wait_event}* parameter indicates information about locks, lightweight locks, and I/Os. Example: -O "19995 kill_wait_event relation" • **"*{port}* kill_business_sql"**: kills all service SQL statements of non-dbadmin and non-rdsAdmin users. There is no parameter. Example: -O "19995 kill_business_sql" • **"*{port}* kill_slow_sql *{username}*"**: kills SQL statements that are executed for more than 1 minute and are not in the idle state in a non-default database in batches. The *{username}* parameter specifies the username. Example: -O "19995 kill_slow_sql omm"

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range

		• "*{port}* **kill_slow_sql** *{username} {database}*": kills SQL statements that are executed for more than 1 minute in a specified database by a specified user in batches. The *{username}* parameter specifies the username, and the *{database}* parameter specifies the database name. Example:
		`-O "19995 kill_slow_sql omm db1"`
		• "*{port}* **kill_executing_sql** *{username}*": kills the SQL statements that are being executed by a specified user whose status is **active** in a non-default database. The *{username}* parameter specifies the username. Example:
		`-O "19995 kill_executing_sql omm"`
		• "*{port}* **kill_mem_topn** *{top n}*": kills the SQL statements that occupy top *n* memory. The *{top n}* parameter specifies the top *n*. Example:
		`-O "19995 kill_mem_topn 10"`
		• "*{port}* **kill_cpu_topn** *{top n}*": kills the SQL statements that occupy top *n* CPU time. The *{top n}* parameter specifies the top *n*. Example:
		`-O "19995 kill_cpu_topn 10"`

- Simple O&M meta-commands of gsql

  For details about how to use the simple O&M meta-commands of gsql, see **Operation Guide**. For details about how to use the meta-commands of gsql, see "Tool Reference > Database Connection Tools > gsql for Connecting to a Database > Meta-Command Reference" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-51 Common simple O&M meta-commands

Simple O&M Command	Description	SQL Statement
\view stmt_track {unique_sql_id}	**stmt_track**: records the level-2 full SQL information of a specified SQL statement. **unique_sql_id**: specifies the unique SQL ID of the SQL statement to be recorded.	SELECT * FROM pg_catalog.dynamic_func_control('LOCAL', 'STMT', 'TRACK', '{"{unique_sql_id}", "L2"}');
\view stmt_untrack {unique_sql_id}	**stmt_untrack**: cancels the recording of the level-2 full SQL information of a specified SQL statement. **unique_sql_id**: specifies the unique SQL ID of the SQL statement whose recording needs to be canceled.	SELECT * FROM pg_catalog.dynamic_func_control('LOCAL', 'STMT', 'UNTRACK', '{"{unique_sql_id}"}');
\view stmt_list	**stmt_list**: queries the record list.	SELECT * FROM pg_catalog.dynamic_func_control('LOCAL', 'STMT', 'LIST', '{}');
\view stmt_clean	**stmt_clean**: clears the record list.	SELECT * FROM pg_catalog.dynamic_func_control('LOCAL', 'STMT', 'CLEAN', '{}');
\view explain_and_stack {pid}	**explain_and_stack**: queries the runtime plan and stack of a specified session. **pid**: specifies the PID of a session.	SELECT pg_catalog.gs_get_explain(pid), pg_catalog.gs_stack(pid) FROM pg_catalog.pg_stat_activity WHERE pid = {pid};
\view slow_sql {top_n}	**slow_sql**: queries real-time slow SQL statements. **top_n**: specifies the number of rows to be queried.	SELECT pid, sessionid, pg_catalog.SUBSTRING(query, 0, 100), state, usename, pg_catalog.NOW() - query_start AS elp FROM pg_catalog.pg_stat_activity WHERE state != 'idle' AND usename NOT IN ('rdsAdmin', 'rdsMetric', 'rdsBackup', 'rdsRepl', 'dbmind_monitor_agent', 'dbmind_monitor') ORDER BY elp DESC LIMIT {top_n};

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view slow_trans {top_n}	**slow_trans**: queries real-time long transactions. **top_n**: specifies the number of rows to be queried.	SELECT pid, sessionid, pg_catalog.SUBSTRING(query, 0, 100), state, usename, pg_catalog.NOW() - xact_start AS elp FROM pg_catalog.pg_stat_activity WHERE state != 'idle' AND usename NOT IN ('rdsAdmin', 'rdsMetric', 'rdsBackup', 'rdsRepl', 'dbmind_monitor_agent', 'dbmind_monitor') ORDER BY elp DESC LIMIT {top_n};
\view wait_event	**wait_event** (without parameters): queries the current statement and wait event.	SELECT a.pid, a.sessionid, a.datname, a.usename, a.client_addr, a.xact_start, a.query_start, (pg_catalog.NOW() - a.query_start)::text "query_runtime", w.wait_status, w.wait_event, w.locktag, w.lockmode, w.block_sessionid, a.query FROM pg_catalog.pg_stat_activity a JOIN pg_catalog.pg_thread_wait_status w ON a.sessionid = w.sessionid WHERE a.pid <> pg_catalog.pg_backend_pid() AND a.state = 'active' AND a.client_addr IS NOT NULL ORDER BY query_runtime DESC;

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view wait_event {unique_sql_id} {start_time} {end_time}	**wait_event** (with three parameters): queries the historical wait events of a specified slow SQL statement. **unique_sql_id**: specifies the unique SQL ID of the slow SQL statement to be queried. **start_time**: specifies the start time. For example, '2025-01-01 00:00'. **end_time**: specifies the end time. For example, '2025-01-01 00:00'.	SELECT *, pg_catalog.statement_detail_decode(details, 'plaintext', true) FROM dbe_perf.statement_history WHERE unique_query_id = {unique_sql_id} AND start_time > {start_time} AND start_time < {end_time} ORDER BY db_time DESC;
\view thr_wait_event	**thr_wait_event** (without parameters): queries real-time overall wait events.	SELECT wait_status, wait_event, pg_catalog.COUNT(*) FROM pg_catalog.pg_thread_wait_status GROUP BY 1, 2 ORDER BY 3 DESC;
\view thr_wait_event {sessionid}	**thr_wait_event** (with one parameter): queries wait events of a specified statement. **sessionid**: specifies the session ID.	SELECT * FROM pg_catalog.pg_thread_wait_status WHERE sessionid = {sessionid};
\view gs_asp {start_time} {end_time}	**gs_asp**: queries the overall wait events in a specified period. **start_time**: specifies the start time. For example, '2025-01-01 00:00'. **end_time**: specifies the end time. For example, '2025-12-31 23:59'.	SELECT wait_status, event, pg_catalog.COUNT(*) FROM pg_catalog.gs_asp WHERE sample_time BETWEEN {start_time} AND {end_time} GROUP BY 1, 2 ORDER BY 3 DESC;
\view pg_stats {table_name} {att_name}	**pg_stats**: queries the distribution of statistics information in a specified table. **table_name**: specifies the table name. **att_name**: specifies the column name.	SELECT * FROM pg_catalog.pg_stats WHERE tablename = '{table_name}' AND attname = '{att_name}';

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view autovac {table_name}	**autovac**: queries the threshold of updating statistics information. **table_name**: specifies the table name.	SELECT * FROM pg_catalog.pg_autovac_status('{table_name}'::regclass);
\view thr_pool	**thr_pool**: queries the thread pool usage.	SELECT node_name, worker_info FROM dbe_perf.global_threadpool_status;
\view failed_proc {proc_name}	**failed_proc**: queries invalid views or stored procedures. **proc_name**: specifies the invalid function.	SELECT re.ev_class, cl.relname FROM pg_catalog.pg_rewrite re, pg_catalog.pg_class cl WHERE re.ev_action LIKE '%{proc_name}%' AND cl.oid = re.ev_class;
\view failed_proc_pkg	**failed_proc_pkg**: queries the package of the invalid stored procedure.	SELECT DISTINCT object_type, pg_catalog.pg_get_userbyid(creator) AS owner, CASE object_type WHEN 'P' THEN (SELECT DISTINCT pg_catalog.UPPER(pkgname) FROM pg_catalog.gs_package WHERE oid = (SELECT propackageid FROM pg_catalog.pg_proc WHERE oid = object_oid)) WHEN 'B' THEN (SELECT DISTINCT pg_catalog.UPPER(pkgname) FROM pg_catalog.gs_package WHERE oid = object_oid) WHEN 'S' THEN (SELECT DISTINCT pg_catalog.UPPER(pkgname) FROM pg_catalog.gs_package WHERE oid = object_oid) END package_name FROM pg_catalog.pg_object WHERE valid = 'f' AND object_type IN ('P', 'B', 'S');

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view failed_index	**failed_index**: queries unavailable indexes.	SELECT a.schemaname, a.relname, a.indexrelname, b.indisusable FROM pg_catalog.pg_stat_user_indexes a, pg_catalog.pg_index b WHERE a.indexrelid = b.indexrelid AND b.indisusable = 'f';
\view table_size	**table_size**: queries the sizes of all tables in the current database and outputs the sizes in descending order.	SELECT schemaname, relname, pg_catalog.pg_total_relation_size(pg_catalog.CONCAT(schemaname, '.', relname)) / 1024 / 1024 / 1024 "GB" FROM pg_catalog.pg_stat_user_tables ORDER BY 3 DESC;
\view dead_tup {top_n}	**dead_tup**: queries the number of dead tuples. **top_n**: specifies the number of rows to be queried.	SELECT relname, n_live_tup, n_dead_tup, (n_dead_tup / (n_live_tup + 1)) AS rating, last_vacuum, last_autovacuum, last_analyze, last_autoanalyze FROM pg_catalog.pg_stat_user_tables ORDER BY rating DESC, n_dead_tup DESC LIMIT {top_n};
\view table_size {table_name}	**table_size**: queries the size of a specified table. **table_name**: specifies the table name.	SELECT * FROM pg_catalog.pg_size_pretty(pg_catalog.pg_total_relation_size('{table_name}'::regclass));

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view login {start_time} {end_time} {ip} {port}	**login**: queries the login and logout information of the specified IP address and port. **start_time**: specifies the start time. For example, '2024-01-01 00:00:00'. **end_time**: specifies the end time. For example, '2025-12-31 23:59:59'. **ip**: specifies the IP address. **port**: specifies the port number.	SELECT *, pg_catalog.SUBSTRING(client_conninfo::varchar, pg_catalog.POSITION(client_conninfo::varchar, '@'::varchar) + 1, 15) AS ip FROM pg_catalog.pg_query_audit({start_time}, {end_time}) WHERE ip = '{ip}' AND remote_port = '{port}';
\view mem	**mem**: queries the current memory usage.	SELECT * FROM pg_catalog.pg_total_memory_detail;
\view thr_mem {top_n}	**thr_mem** (with one parameter): queries the memory context usage of all database threads. **top_n**: specifies the number of rows to be queried.	SELECT contextname, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize", pg_catalog.SUM(freesize) / 1024 / 1024 "freesize", pg_catalog.COUNT(*) "sum" FROM pg_catalog.gs_thread_memory_context GROUP BY contextname ORDER BY sum DESC LIMIT {top_n};
\view thr_mem {context_name} {top_n}	**thr_mem** (with two parameters): queries the thread ID of the thread that uses the most memory. **context_name**: specifies the name of the memory context. **top_n**: specifies the number of rows to be queried.	SELECT * FROM pg_catalog.pv_thread_memory_detail() WHERE contextname = '{context_name}' ORDER BY totalsize DESC LIMIT {top_n};

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view sess_mem {top_n}	**sess_mem**: queries the memory context usage of all sessions in the database. **top_n**: specifies the number of rows to be queried.	SELECT contextname, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize", pg_catalog.SUM(freesize) / 1024 / 1024 "freesize", pg_catalog.COUNT(*) "sum" FROM pg_catalog.gs_session_memory_context GROUP BY contextname ORDER BY sum DESC LIMIT {top_n};
\view thr_mem_detail {tid} {context_name}	**thr_mem_detail**: queries the memory usage of a specified thread. **tid**: specifies the thread ID. **context_name**: specifies the name of the memory context.	SELECT * FROM pg_catalog.gs_get_thread_memctx_detail({tid}, '{context_name}');
\view sess_mem_tp {context_name}	**sess_mem_tp**: queries the memory usage of a specified memory context. **context_name**: specifies the name of the memory context.	SELECT * FROM pg_catalog.gs_get_session_memctx_detail('{context_name}');
\view shared_mem {top_n}	**shared_mem**: queries the usage of the global shared memory context in the database. **top_n**: specifies the number of rows to be queried.	SELECT contextname, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize MB", pg_catalog.SUM(freesize) / 1024 / 1024 "freesize MB", pg_catalog.COUNT(*) "sum" FROM pg_catalog.gs_shared_memory_detail GROUP BY contextname ORDER BY sum DESC LIMIT {top_n};
\view sess_mem_detail {top_n}	**sess_mem_detail**: queries the usage of a session-level memory context. **top_n**: specifies the number of rows to be queried.	SELECT contextname, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize MB", pg_catalog.COUNT(1) "count" FROM pg_catalog.pv_session_memory_detail() GROUP BY contextname ORDER BY 2 DESC LIMIT {top_n};

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\view sess_mem_detail_tp {top_n}	**sess_mem_detail_tp**: queries the usage of a session-level memory context (valid only when the thread pool mode is enabled). **top_n**: specifies the number of rows to be queried.	SELECT sessid, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize MB", pg_catalog.COUNT(1) "count" FROM pg_catalog.gs_session_memory_context GROUP BY sessid ORDER BY 2 DESC LIMIT {top_n};
\view thr_mem_pv {top_n}	**thr_mem_pv**: queries the usage of a thread-level memory context. **top_n**: specifies the number of rows to be queried.	SELECT contextname, pg_catalog.SUM(totalsize) / 1024 / 1024 "totalsize", pg_catalog.SUM(freesize) / 1024 / 1024 "freesize", pg_catalog.COUNT(1) "count" FROM pg_catalog.pv_thread_memory_detail() GROUP BY 1 ORDER BY 2 DESC LIMIT {top_n};
\view total_mem {top_n}	**total_mem**: queries the memory usage of the current database node. **top_n**: specifies the number of rows to be queried.	SELECT * FROM pg_catalog.gs_total_memory_detail LIMIT {top_n};
\view avg_cpu {top_n}	**avg_cpu**: queries the average CPU time of SQL statements and outputs the query result in descending order. **top_n**: specifies the number of rows to be queried.	SELECT unique_sql_id, cpu_time / (n_calls) "average_cpu_time" FROM dbe_perf.statement WHERE n_calls != 0 ORDER BY 2 DESC LIMIT {top_n};
\om create_abort_sql_patch {patch_name} {unique_sql_id}	**create_abort_sql_patch**: creates an abort SQL patch. **patch_name**: specifies the SQL patch name. **unique_sql_id**: specifies the unique SQL ID.	SELECT * FROM dbe_sql_util.create_abort_sql_patch('{patch_name}', {unique_sql_id});

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Simple O&M Command	Description	SQL Statement
\om kill_sess_thr {username} {database} {context_name}	**kill_sess_thr**: kills SQL statements that occupy a large amount of memory based on the thread memory usage. **username**: specifies the username. **database**: specifies the database name. **context_name**: specifies the name of the memory context.	SELECT pg_catalog.pg_terminate_session(pid, sessionid), pid, sessionid FROM pg_catalog.pg_stat_activity WHERE usename = '{username}' AND datname != 'postgres' AND datname = '{database}' AND pid IN (SELECT tid FROM pg_catalog.gs_thread_memory_context WHERE contextname = '{context_name}') AND state = 'idle';
\om kill_sess_sess {username} {database} {context_name}	**kill_sess_sess**: kills SQL statements that occupy a large amount of memory based on session memory usage. **username**: specifies the username. **database**: specifies the database name. **context_name**: specifies the name of the memory context.	SELECT pg_catalog.pg_terminate_session(pid, sessionid), pid, sessionid FROM pg_catalog.pg_stat_activity WHERE usename = '{username}' AND datname != 'postgres' AND datname = '{database}' AND pid IN (SELECT threadid FROM pg_catalog.gs_session_memory_context WHERE contextname = '{context_name}');

- Emergency channel view DBE_PERF.EMERGENCY_OPERATION_RECORDS

  This view displays the usage of the UDS O&M channel on each CN or DN.

  Table 7-52 DBE_PERF.EMERGENCY_OPERATION_RECORDS columns

Parameter	Type	Description
record_time	timestamptz	Time when the O&M channel is used.
action	text	Operations performed through the O&M channel.
source	text	Source of the O&M channel. The value can be **gsql** or **cm**.
detail	text	Operation result.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.29.3 Operation Guide

Prerequisites

The GUC parameter **enable_emergency_access** must be enabled for the emergency channel. In addition, the O&M capability of generating **mem_log** depends on the wlm_monitor thread and memory management functionality. Therefore, the GUC parameters **use_workload_manager** and **enable_memory_limit** must be enabled. By default, they are enabled.

For details about the GUC parameter **enable_emergency_access**, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

For details about the GUC parameter **use_workload_manager**, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

For details about the GUC parameter **enable_memory_limit**, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*.

Example

Example 1

Run an O&M command through the UDS O&M channel.

The following is a simple example of how to check the **mem_log** generated through the UDS O&M channel:

Step 1 Log in to the server where the GaussDB node is located and run the following gsql command to check whether the memory usage records, such as memory overview and memory context, are generated:

```
gsql -O "{port} mem_log"
```

{port} is a database port. If **true** is displayed, the detailed information is generated. If the execution fails, the failure cause is displayed.

```
-- In the example, 19995 is the database listening port.
gsql -O "19995 mem_log"
true
```

Step 2 Check the content of the **mem_log** file generated in the *$GAUSSLOG*/**mem_log/***{nodename}* directory. *{nodename}* specifies the database node name.

```
-- In the example, datanode specifies the database node name.
ls -l $GAUSSLOG/mem_log/datanode
total 8
-rw-------. 1 omm omm 7543 May 28 15:15 mem_log-2025-05-28_151518.log
```

Step 3 Run the following command to view the file:

```
cat $GAUSSLOG/mem_log/datanode/mem_log-time.log
```

mem_log-time.log is the file generated in the *$GAUSSLOG*/**mem_log/***datanode* directory in Step 2. (The file name extension is .log.)

----End

Example 2

Execute the simple O&M meta-command **\view mem** of gsql.

```
-- Connect to a database. (Replace the database name and port number as required.)
gsql -d postgres -p 8000
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gsql ((GaussDB Kernel XXX.X.XXX build f521c606) compiled at 2025-08-30 11:51:09 last mr 24890 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
gaussdb=# \view mem
 nodename |            memorytype            | memorymbytes
----------+----------------------------------+--------------
 cn_5001  | max_process_memory               |        35840
 cn_5001  | process_used_memory              |        12417
 cn_5001  | max_dynamic_memory               |        28532
 cn_5001  | dynamic_used_memory              |         3448
 cn_5001  | dynamic_peak_memory              |         8966
 cn_5001  | dynamic_used_shrctx              |         2081
 cn_5001  | dynamic_peak_shrctx              |         2815
 cn_5001  | max_backend_memory               |         1544
 cn_5001  | backend_used_memory              |            1
 cn_5001  | high_backend_thread_memory       |           56
 cn_5001  | high_backend_thread_max_memory   |          921
 cn_5001  | medium_backend_thread_memory     |          358
 cn_5001  | medium_backend_thread_max_memory |         3072
 cn_5001  | low_backend_thread_memory        |            0
 cn_5001  | low_backend_thread_max_memory    |         1536
 cn_5001  | max_shared_memory                |         5747
 cn_5001  | shared_used_memory               |         4866
 cn_5001  | max_cstore_memory                |           16
 cn_5001  | cstore_used_memory               |            0
 cn_5001  | max_sctpcomm_memory              |            0
 cn_5001  | sctpcomm_used_memory             |            0
 cn_5001  | sctpcomm_peak_memory             |            0
 cn_5001  | other_used_memory                |         3998
 cn_5001  | gpu_max_dynamic_memory           |            0
 cn_5001  | gpu_dynamic_used_memory          |            0
 cn_5001  | gpu_dynamic_peak_memory          |            0
 cn_5001  | pooler_conn_memory               |            1
 cn_5001  | pooler_freeconn_memory           |            0
 cn_5001  | storage_compress_memory          |            0
 cn_5001  | udf_reserved_memory              |            0
 cn_5001  | llvm_used_memory                 |            9
 cn_5001  | max_htap_memory                  |            0
 cn_5001  | htap_used_memory                 |            0
(33 rows)
```

7.29.4 FAQ

1. **Symptom**: After the gsql -O "{port} mem_log" command is executed, the mem_log file is not generated.

   Solution: The O&M capability of generating **mem_log** depends on the wlm_monitor thread and memory management functionality, which are available when the **use_workload_manager** and **enable_memory_limit** parameters are enabled. Therefore, check whether they have been enabled.

   ```
   gaussdb=# SHOW use_workload_manager;
    use_workload_manager
   ----------------------
    on
   (1 row)

   gaussdb=# SHOW enable_memory_limit;
    enable_memory_limit
   ---------------------
    on
   (1 row)
   ```

   For details about how to enable the **use_workload_manager** parameter, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. For details about how to enable the **enable_memory_limit** parameter, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. **Symptom**: O&M commands of the emergency channel cannot be executed on non-database nodes.

   Solution: This functionality supports only local connections and needs to be used on database nodes.

7.30 CM Arbitration and Fault Detection Principles

7.30.1 Overview

The CM module consists of the CM Server and CM Agent.

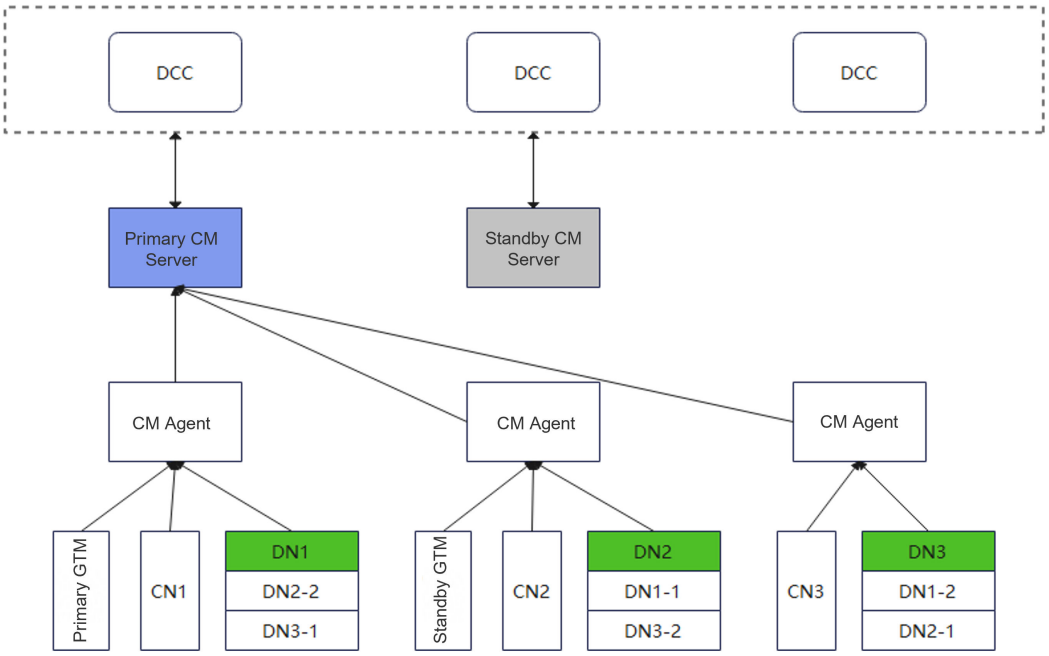
CM Server acts as the brain of the cluster. It monitors the status of all GTM, CN, and DN instances and manages HA arbitration for primary/standby GTMs and DNs. By analyzing node status reported by CM Agent, CM Server determines if and how an instance's state should be transitioned. Under normal conditions, the cluster must have one and only one active CM Server to perform arbitration. The CM Server periodically writes heartbeats to ETCD. If the active CM Server fails, a new primary is elected through ETCD-based arbitration.

CM Agent provides the following functions: starting and stopping all instances on the local node, monitoring and reporting the status of all instances on the local node to CM Server, and executing arbitration commands delivered by CM Server.

CM Agent is a multi-thread monitoring component. **Figure 7-58** shows the thread structure. The threads include:

1. Main thread: reports the status of node instances and executes commands delivered by CM Server.
2. Start/Stop thread: starts and stops node instances.
3. GTM/DN/CN connection thread: queries the instance status.

Figure 7-58 CM Agent thread structure



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.30.2 Technical Principles

7.30.2.1 Arbitration Principle

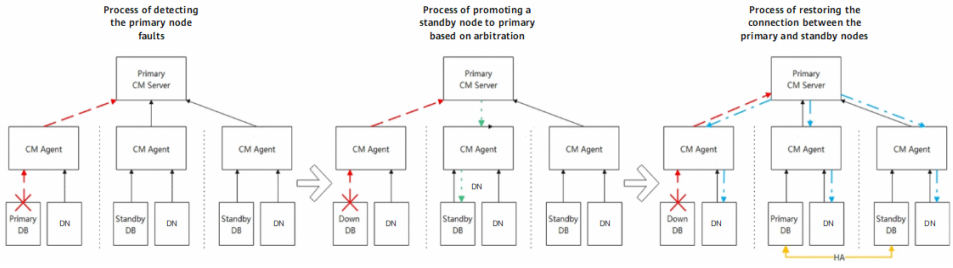
7.30.2.1.1 DN Arbitration

CM supports two DN arbitration modes: quorum (default) and DCF.

The following figure shows the DN arbitration process for leader election in quorum mode.

1. The primary DN is faulty (heartbeat timeout: 6s/process down/disk fault/port fault/zombie state).
2. The CM detects that no primary DN exists and sends the **lock1** command to lock normal standby DNs. If redo logs are replayed and the primary and standby DNs are disconnected, **lock1** successfully converts the standby DNs to the prohibit_connection state, indicating that they will not accept new logs and the local logs have been replayed. The CM can obtain the largest local log on the DNs.
3. The standby DN with the largest local log is selected from the majority of locked standby DNs and instructed to become the primary.
4. The CM instructs the CN to update pgxc_node, which contains the routing information of the primary DN. A CN connects to the primary DN through pgxc_node.
5. The CM sends the **lock2** command to the standby DNs. With the routing information of the new primary DN in the command, the standby DNs can connect to the new primary DN.
6. The CM initiates the **unlock** command to restore the free connection status of the primary and standby DNs.
7. The primary/standby relationship in ETCD is updated, where **1** indicates the primary, and **2** indicates the standby.
8. The services are recovered

Figure 7-59 DN arbitration for leader election in quorum mode



In the multi-AZ scenario, the principle for handling the primary DN fault is as follows:

If the primary DN is faulty, the standby DN that is initially specified by the user, has the latest data, is in the same AZ as the original primary DN, and has the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

highest AZ priority (configured during installation) is preferentially promoted to primary in descending order of priority. After the primary DN that has been demoted to standby recovers, it cannot be automatically switched back to primary but requires manual switchover. If possible, the logs on original primary DN catch up with those on the new primary DN. Otherwise, full data rebuild is performed on the original primary DN.

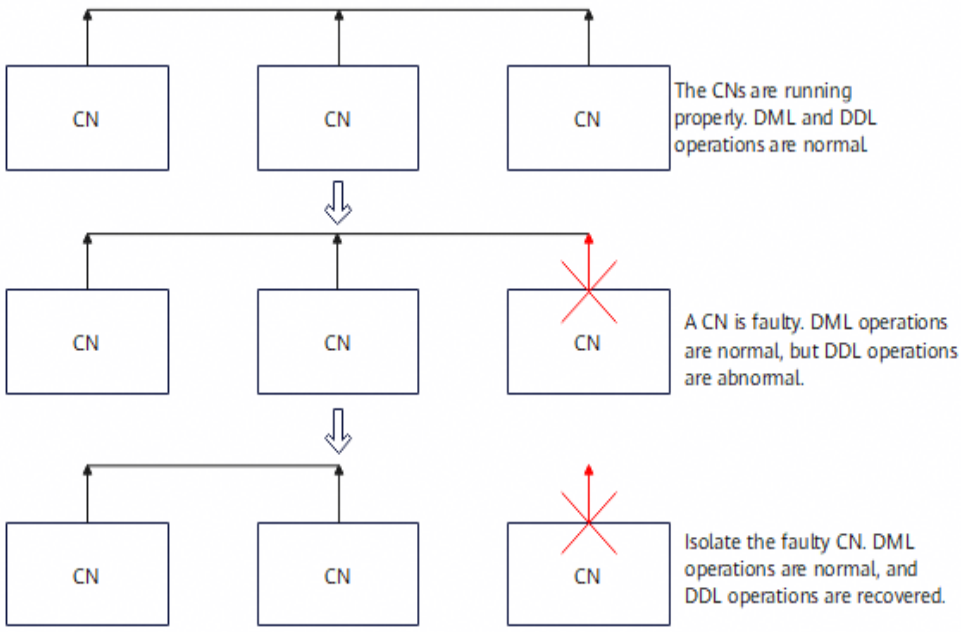
7.30.2.1.2 CN Arbitration

CNs are not deployed in primary/standby mode. All DDL operations on a CN are synchronously performed on all other CNs with strong consistency. If the CN is faulty, DDL operations will be affected. If the CN still cannot recover over 25s after the fault occurs, the CM considers that the CN cannot provide services and starts the CN removal process. The following figure shows the CN removal process.

CN removal:

1. When the CM detects that the CN is faulty (the process is down, in the zombie state, or restarted for multiple times, the disk is faulty, port conflicts exist, the NIC is faulty, or the CN cannot connect to all primary DNs) for more than 25s, the CM initiates CN removal. The CN is in the **Deleting** state, and the value in ETCD is updated to **8**.
2. The CM updates the pgxc_node system catalog of other nodes. The CN is in the **Deleted** state, and the value in ETCD is updated to **7**.
3. The database status is recovered, and the CN can perform DDL operations.

Figure 7-60 CN removal process



The process of resuming a CN is reverse. The CN can be recovered if the following conditions are met:

1. The CN fault is rectified, meeting the conditions for resuming the CN.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. The build process synchronizes CN logs and updates pgxc_node.
3. The CN status is set to **Normal**, and the value in ETCD is updated to **4**.

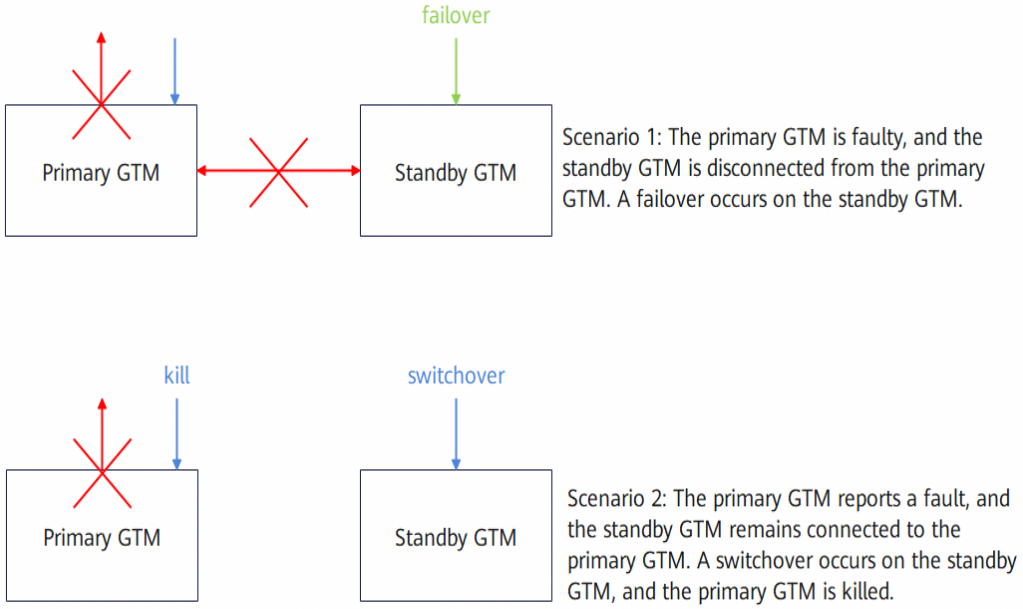
7.30.2.1.3 GTM Arbitration

GTM is short for global transaction manager. There is only one GTM in the system, which works in primary/standby mode. If the primary GTM is faulty, the standby GTM can take over services. The primary GTM does not synchronize data with the standby GTM. The standby GTM only attempts to connect to the primary GTM to verify the connection status. The data to be synchronized between the primary and standby GTMs is backed up on ETCD. The following figure shows the GTM arbitration process for leader election.

GTM arbitration for leader election:

1. The primary GTM is faulty (the process is down, the disk is faulty, port conflicts exist, or the NIC is faulty).
2. The system checks the connection status between the primary and standby GTMs.
3. If the standby GTM is disconnected from the primary GTM, the standby GTM reported as normal is promoted to primary through a failover.
4. If the standby GTM is connected to the primary GTM, a primary/standby switchover is triggered and the original primary GTM process is terminated.
5. After the switchover is complete, the CN is instructed to update the GTM connection.

Figure 7-61 GTM arbitration for leader election



7.30.2.1.4 CM Server Arbitration

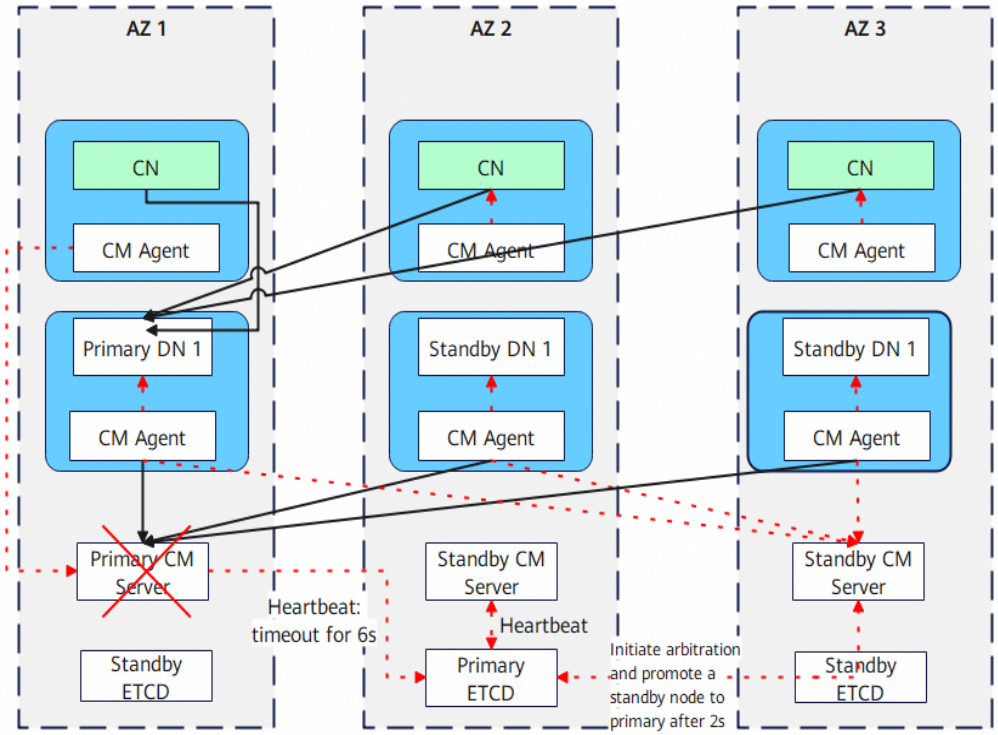
CM Server is the brain of a cluster. It arbitrates instances in the cluster and delivers commands. When an instance is faulty, CM Server arbitrates the instances to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

restore the cluster. However, when CM Server is faulty, there is no upper-layer component for arbitration. CM Server uses its own arbitration mechanism to ensure the availability of the cluster.

CM Server automatic arbitration and leader election:

1. CM Server continuously writes heartbeats to ETCD.
2. If the primary CM Server is abnormal, it does not update heartbeats to ETCD. After the standby CM Server detects that the heartbeat timeout of the primary CM Server reaches 6s, the standby CM Server enters the automatic leader election process.
3. If the standby CM Server meets the heartbeat continuity requirement and has a pre-connection to CM Agent, the standby CM Server automatically becomes the primary and writes the information about the promotion to ETCD. Other CM Servers exit the leader election process after detecting the information about the promotion in ETCD.
4. The primary CM Server is successfully elected.



7.30.2.1.5 ETCD Arbitration

ETCD is an open source distributed key-value store used for configuration sharing and service discovery. It stores the topology information of clusters, provides APIs for storing and obtaining data, and uses protocols to ensure strong consistency of data among multiple ETCD cluster nodes. ETCD can be used to store metadata and share configuration information. It also offers a listening mechanism that allows clients to listen on and push the changes of one or more specific keys in real time. In addition, the key expiration and renewal mechanisms are available in ETCD. Clients can periodically update keys to renew them. This is applicable to monitoring and service registry and discovery. Furthermore, ETCD supports atomic

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Compare-and-Swap (CAS) and Compare-and-Delete (CAD), distributed locks, and leader election.

Functionalities of ETCD:

1. Storing global transaction IDs and sequences depends on ETCD.
2. CM Server's arbitration of itself and other database components depend on ETCD.
3. The storage of primary/standby information of each component depends on ETCD.

ETCD arbitration for leader election:

1. ETCD automatic leader election is based on the Raft distributed consensus algorithm, which ensures data consistency among all nodes in a cluster.
2. If the primary ETCD node (leader) is faulty, the standby ETCD nodes (followers) initiate a vote. The candidate that obtains the majority of votes becomes the primary node. Then, the new primary ETCD node notifies the primary CM Server node of the promotion. The primary CM Server node and the GTM synchronizes data with the new primary ETCD node.

7.30.2.2 Fault Detection Principle

7.30.2.2.1 Disk Check

- Specifications:
  - Only the data disks and log disks of DNs, CNs, GTMs, and CMs (CM Agent and CM Server) and the disks where DN tablespaces reside are detected.
- Detection rules
  - Detection is implemented by reading and writing temporary files. There are four operations on files: open (opening a file), write (writing a file), read (reading a file), and unlink (deleting a file). If any of the open, write, and read operations fails or times out, the system determines that the disk is faulty or hung and triggers arbitration. On the contrary, if the unlink operation fails or times out, arbitration is not triggered.
  - Non-direct read/write: 1-byte data is written to the disk in non-direct mode every second. If the read/write fails, determine the failure cause based on the error code. If any of the EIO (I/O error), EROFS (read-only file system), ENOSPC (insufficient disk space), and EACCES (insufficient permission) error codes is found, a fault is reported for handling. Other errors are not handled.
  - Direct read/write: By default, 512 bytes (or 256 KB if the detected disk is striped) data is written to the disk in direct mode every 5 seconds (configurable). If the write operation is blocked for more than 20 seconds (or 60s assuming that the size of the written data is 256 KB if you cannot determine whether the detected disk is striped) and no result is returned, the disk is in I/O hang state.
- Disk hang under heavy load
  - Introduction: The CM periodically obtains physical I/O indicators (number of file read/write times and file read/write traffic) from the database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

After detecting a disk hang, the CM first determines if the I/O indicator meets the high-load standard (10 IOPS or 256 KB of file read/write traffic per second). If it reaches the high-load standard, the CM will not proceed with further processing of the disk hang.

  - Limitations:
    - High-load scenario definition: The system is considered to be in a high-load state when the number of file reads/writes reaches 10 IOPS or the file read/write traffic reaches 256 KB per second.
    - In high-load scenarios, disk hang processing is not performed. In non-high-load scenarios, disk hang processing is performed, and the RTO is less than 60s.
    - If the I/O indicator cannot be obtained, the disk hang is processed and the RTO is less than 60s.
- Note:
  - The size of the data to be written is configurable. The configured size must be aligned based on the disk alignment rule. If the size is not aligned, the CM automatically aligns the size in a roundup mode. In disk LVM striping scenarios, to ensure that write detection covers all disks, you are advised to dynamically adjust the write size based on the number of disks and the size of data blocks.
  - The I/O hang timeout interval is configurable and needs to be configured based on disk performance. For disks with poor performance, the timeout interval can be prolonged to reduce the impact of service I/Os on disk detection. Accordingly, the RTO is prolonged in hang scenarios. If it is set to **0**, hang detection is disabled.
  - If the write size is too large or the detection period is too short, the contention of disk I/O resources may occur, which may further affect the database performance.

7.30.2.2.2 Zombie DN Detection

When a DN process is in the **D**, **T**, **Z**, or connection timeout state, the DN cannot process service requests promptly. As a result, services are interrupted and service functions are affected. The CM periodically checks whether a DN is in the zombie state based on the configured value.If a DN is in the zombie state for a specified number of consecutive times, the DN will be automatically restarted for recovery.

7.30.2.2.3 First Aid Capability

First aid is a highly abstract framework model. False positives and negatives may occur due to faults. Therefore, the framework needs to be more flexible and available for different scenarios and requirements. Thread heartbeat self-detection, disk I/O hang (data cannot be written or read due to disk faults), automatic primary DN isolation, system resource monitoring, network detection, and asynchronous log functions are supported.

7.30.2.2.4 Hardware Troubleshooting

Hardware troubleshooting is an end-to-end full-stack HA capability.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.30.3 Operation Guide

Arbitration and fault detection are automatically performed.

7.30.4 FAQ

Q: Why is a CN not added back after the fault is rectified?

A: Search for **autorepair** in the CM Agent logs of the corresponding node to check the error cause. After the fault is rectified, if the CM does not add the CN back within 30 minutes, it will not attempt to add the CN back again. In this case, you can add back the CN by replacing the node.

Q: After a majority of nodes are faulty, the cluster has no primary node.

A: For details, see "Best Practices for Handling a Majority Fault" in *Best Practices*.

Q: A primary/standby DN switchover occurs.

A: Search for **killing xxx by force** and **inst(xxx) data(xxx) killed** in the CM Agent logs of the original primary node, or search for **KEY_EVENT_RESTART** in the primary CM Server logs. If related entries exist, the primary/standby switchover is triggered by CM arbitration. Then, check whether keywords such as **io hang** or **phony dead** exist around the time when the process is terminated to determine the specific cause of the primary/standby switchover. If related CM arbitration records are not found, the fault is not caused by CM arbitration. In this case, further check the cause of the DN process exit.

Q: The RTO of the primary/standby failover times out after a fault occurs.

A: Search for **KEY_EVENT_FAILOVER** in the key_event_xxx log of the active CM Server node. The timestamp preceding this event indicates the CM arbitration and leader election time, while the timestamp following it represents the duration or completion of the DN promotion to primary.

1. If the arbitration latency is high, examine the active CM Server and CM Agent logs to pinpoint when the fault was detected and when the decision was made. Check for cascading failures, such as concurrent failures of the ETCD leader, active CM Server, and primary DN.
2. If the DN promotion is delayed, analyze the logs of the DN being promoted to identify the root cause of the slow failover.

7.31 Principles of Intra-cluster and Inter-cluster Switchover

7.31.1 Intra-cluster Switchover

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.31.1.1 Overview

Description

This feature implements planned switchover between the primary and standby DNs in a cluster to ensure service continuity.

Scenarios

The intra-cluster switchover process is involved in cluster installation, upgrade, scale-out, and so on.

Specifications and Limitations

- If the primary DN has no services and the standby DN has no stacked logs (the RTO of the standby DN is less than 10 seconds), the switchover timeout interval is 120 seconds.
- Before performing a switchover, check the RTO of the standby DN that will be promoted to primary. If the RTO is large but a switchover is required, you need to manually specify the timeout interval (120 seconds by default).
- A cascaded standby DN cannot be directly converted to a primary DN. It must first be switched or failed over to become a standby DN before it can be promoted to a primary DN.

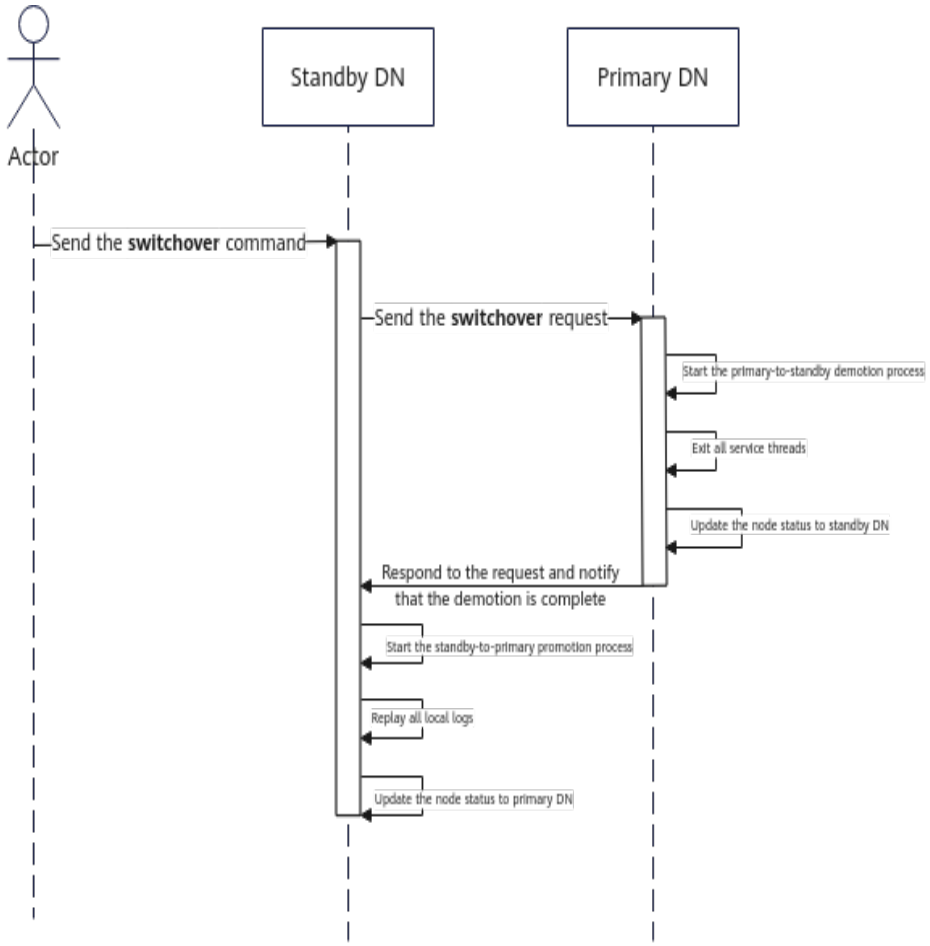
7.31.1.2 Technical Principle

The switchover process is divided into the following core phases:

1. After the **switchover** command is delivered using cm_ctl, the standby DN sends a request to the primary DN for demoting it to standby.
2. After receiving the request, the primary DN performs relevant operations such as exiting all worker threads and destroying the shared memory.
3. The primary DN updates its status to standby and notifies the standby DN that the demotion is complete.
4. After receiving the message, the standby DN replays local logs and updates its status to primary.
5. The switchover process is complete.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-62 Principle of intra-cluster switchover



7.31.1.3 Operation Guide

Prerequisites

Before a switchover, ensure that the database is running properly, all services are stopped, and the pg_get_senders_catchup_time view shows no ongoing catchup between the primary and standby nodes.

Precautions

Before the switchover, run the **CHECKPOINT** command to ensure that dirty pages are flushed to disks and prevent the switchover from timeout.

Examples

- Switch over a specified AZ.
  ```
  cm_ctl switchover -z AZ1
  ```
- Switch over a database instance.
  ```
  cm_ctl switchover -n 1 -D /data1/omm/cluster/data/datanode1p
  ```
- Switch over all GTMs and DNs and promote another instance of the corresponding shard to primary.
  ```
  cm_ctl switchover -A
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Reset the primary/standby relationship of the cluster to the initial configuration status.
  ```
  cm_ctl switchover -a
  ```
- Perform CM Server switchover in DCC mode.
  ```
  cm_ctl switchover -n 1 -D /data1/omm/cluster/cm/cm_server
  ```

7.31.1.4 Typical Issues

- The switchover times out because the primary-to-standby demotion times out.

  The timeout of demoting the primary node to standby is mainly caused by the timeout of exiting background threads or key threads. You need to view logs to locate the fault (*$GAUSSLOG*/**gs_log/dn_xxxx/gaussdb-*.log**).
- The switchover times out due to the timeout of promoting the standby node to primary:
  - Check whether Xlogs are stacked.
  - Check whether error information is recorded in logs during the promotion and determine whether the promotion process is interrupted due to an exception.

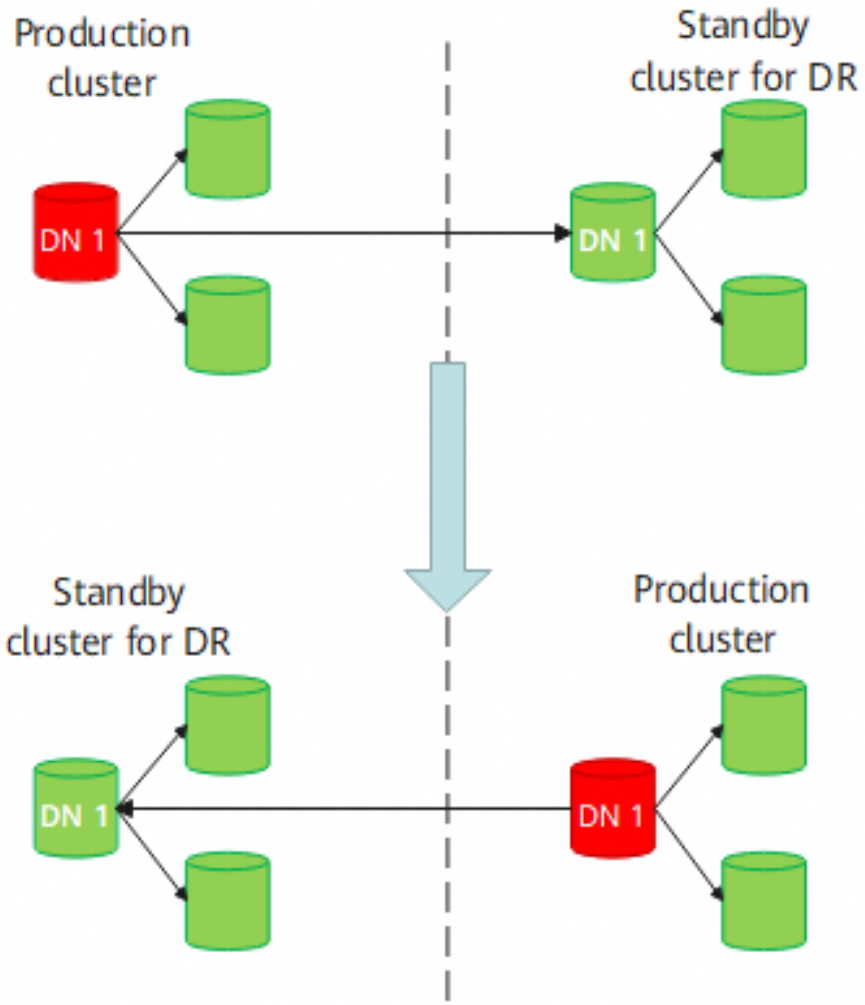
7.31.2 Inter-cluster Switchover

7.31.2.1 Overview

Description

On the premise of zero data loss, the primary and standby database instances are switched over as planned to ensure system availability in scenarios such as database maintenance, upgrade, and DR drills. The following figure shows the switchover principle.

Figure 7-63 Principle of inter-cluster switchover



Scenarios

The DR capability of the core transaction system of a bank needs to be verified periodically. For this purpose, a switchover is performed to switch the primary cluster to the intra-city or remote standby cluster for DR to verify that the RPO is 0 (zero data loss) and the RTO (service recovery time) meets the specifications.

Specifications and Limitations

The specifications and limitations vary depending on the DR mode.

- Streaming replication-based remote DR solution.
  - Paxos protocol–based mode:
    - No logs are stacked (that is, RPO ≤ 10s and RTO ≤ 1s), the number of sequences is less than 10, the primary and standby cluster instances are in the normal state, all components (CN, DN, ETCD, GTM, CM Agent, and CM Server) are normal, and the quick switchover mode is enabled, a DR switchover can be completed within 30s of the kernel time. Especially, if the primary and standby

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

clusters are heterogeneous and the number of CNs is inconsistent, the cluster with many CNs exits in the degraded state after the switchover is complete.

- No logs are stacked (that is, RPO ≤ 10s and RTO ≤ 1s), the number of sequences is less than 10, and some components (CN, DN, ETCD, GTM, CM Agent, and CM Server) of the primary and standby cluster instances are abnormal and meet the requirements of minority faults, if the fast switchover mode is enabled, the DR switchover can be completed within 90s of the kernel time.
- If logs are stacked (that is, RPO > 10s and RTO > 1s), log transmission and replay increase the DR switchover duration. As a result, a switchover within the specifications cannot be implemented.
- On the Arm platform, if the value of **PAGESIZE** (queried by running **getconf PAGESIZE**) does not meet the 64 KB specification, a switchover within the specification cannot be implemented.

– Quorum protocol–based mode:

- No logs are stacked (that is, RPO ≤ 10s and RTO ≤ 1s), the number of sequences is less than 10, the primary and standby cluster instances are in the normal state, all components (CN, DN, ETCD, GTM, CM Agent, and CM Server) are normal, and the quick switchover mode is enabled, a DR switchover can be completed within 20s of the kernel time. Especially, if the primary and standby clusters are heterogeneous and the number of CNs is inconsistent, the cluster with many CNs exits in the degraded state after the switchover is complete.
- No logs are stacked (that is, RPO ≤ 10s and RTO ≤ 1s), the number of sequences is less than 10, and some components (CN, DN, ETCD, GTM, CM Agent, and CM Server) of the primary and standby cluster instances are abnormal and meet the requirements of minority faults, if the fast switchover mode is enabled, the DR switchover can be completed within 90s of the kernel time.
- If logs are stacked (that is, RPO > 10s and RTO > 1s), log transmission and replay increase the DR switchover duration. As a result, a switchover within the specifications cannot be implemented.
- On the Arm platform, if the value of **PAGESIZE** (queried by running **getconf PAGESIZE**) does not meet the 64 KB specification, a switchover within the specification cannot be implemented.

- Intra-city dual-DC HA solution.
  – When the standby cluster can catch up with the primary cluster during replay and the network of the standby cluster is normal, if all clusters are normal, the average RTO for the kernel to perform a failover is less than 30s; if minority faults occur in the clusters, the average RTO for the kernel to perform failover is less than 90s. (The specific time is affected by the number of logs to be replayed in the cluster. The data is configured on 3 CNs, 3 DNs, and 3 copies.)
  – When the standby cluster can catch up with the primary cluster during replay and the network of the standby cluster is normal, if no log is

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

stacked (that is, the DR monitoring indicator RTO is less than or equal to 1s), the number of sequences is less than 10, and all clusters are normal, the average RTO for the kernel to perform switchover is less than 1 minute; if minority faults occur in the clusters, the average RTO for the kernel to perform a switchover is less than 2 minutes. (The specific time is affected by the number of logs to be replayed in the cluster. The data is configured on 3 CNs, 3 DNs, and 3 copies.)

- If the specifications are exceeded, for example, if the service pressure is high for a short period of time, the RTO will inevitably increase. According to the test result, to meet the preceding RTO limitations on failover and switchover in the current version, the log generation rate of a single shard in the primary cluster must be within a specific range. **Table 7-53** lists the log generation rate requirements in typical configurations.

Table 7-53 Log generation speed in typical configurations

Typical Configuration	Single-Shard Log Generation Speed of the Primary Cluster
96U/768G/SATA SSD	<=7.5MB/s

For details about the evolution of the RTO specifications for inter-cluster switchover, see **Table 7-54**.

Table 7-54 Evolution of the RTO specifications for inter-cluster switchover

DR Mode	Deployment Mode	R2C10	503.1.0	503.1.0 SPC2000c	503.2.0	505.0.0	505.1.0	505.2.0	505.2.1 SPC0100	506.0.0	506.1.0 and later
Streaming replication-based remote DR solution	Centralized	5 minutes (reduced to 50s in versions later than SPC660)	5 minutes (reduced to 50s in versions later than SPC1700)	50s	50s	50s	50s	50s	50s	10s	10s

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

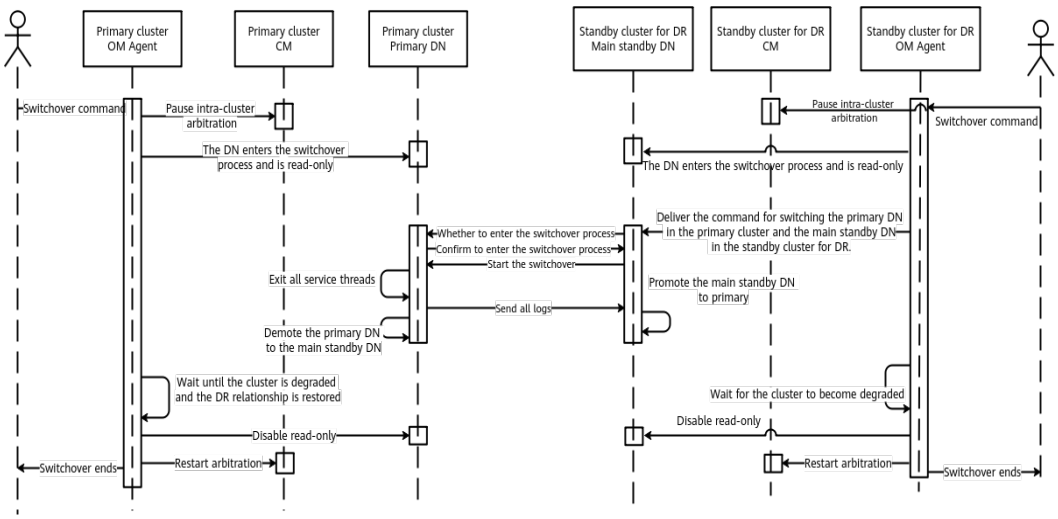
DR Mode	Deployment Mode	R2C10	503.1.0	503.1.0 SPC2000c	503.2.0	505.0.0	505.1.0	505.2.0	505.2.1 SPC0100	506.0.0	506.1.0 and later
	Distributed	10 minutes	10 minutes	10 minutes	10 minutes	60s	60s	30s	30s	20s	20s
Intra-city dual-DC HA solution	Centralized	2 minutes	2 minutes	60s	2 minutes	2 minutes	2 minutes	2 minutes	60s	60s	60s
	Distributed	Not supported	Not supported	Not supported	Not supported	3 minutes	2 minutes	2 minutes	2 minutes	60s	60s

7.31.2.2 Technical Principle

1. After a user delivers a cluster switchover command on the management and control platform, the platform sends a switchover request to the primary and standby clusters at the same time.
2. After receiving the switchover command, the standby cluster for DR suspends the arbitration in the cluster and renders the main standby DN as read-only.
3. The main standby DN in the standby cluster for DR sends a switchover request to the primary DN in the primary cluster through the data transmission link between the primary and standby clusters, and receives incremental logs for replay.
4. After the replay is complete, the standby cluster for DR is promoted to primary.
5. After receiving the switchover request from the standby cluster for DR, the primary cluster exits all worker threads, clears related resources, and demotes to standby.
6. After the roles of the primary and standby clusters are updated, OM Agent disables the DN read-only mode and restarts CM arbitration to complete the switchover.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-64 Inter-cluster switchover process



API Description

1. In the streaming replication-based remote DR solution, the system sends a planned switchover request to the primary and standby database instances through the OM Agent URI /v1/om/agent/mgr/cluster/streaming_disaster_recovery_switchover.
2. In the intra-city dual-DC HA solution, the system sends a request to the database instance to demote to the standby database instance for DR through the OM Agent URI /v1/om/agent/mgr/cluster/dorado_switch.

7.31.2.3 Operation Guide

- For details about the streaming replication-based remote DR solution, see **Streaming Replication–based Remote DR Drill Practice Reference**.
- For details about the intra-city dual-DC HA solution, see **Intra-City Dual-DC HA Solution DR Drill Practice Reference**.

NOTICE

Inter-cluster switchover is a high-risk operation that affects services. It is advisable to perform this operation through the database management platform in accordance with its official guidelines.

7.31.2.3.1 Streaming Replication–based Remote DR Drill Practice Reference

Introduction to DR Drills

The planned switchover function can be used to perform DR drill. This operation enables the roles of the primary and standby clusters to be exchanged. The original standby cluster becomes the primary cluster to provide services, and the original primary cluster becomes the standby cluster. This function ensures that no data is lost, that is, RPO is 0.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-55 Planned migration in the DR drill

Step	Operation	Description
1	**Pre-Check Items**	Check the prerequisites for DR switchover.
2	**Performing a Switchover**	Call the planned switchover API to perform a switchover for primary and standby clusters.
3	**Performing a Post-check After a DR Drill**	After the DR switchover is complete, check the database cluster status, database cluster type, and DR status, and report the IDs of instances that are not involved in the DR switchover. Use the related information to check the process execution result and determine the repair policy.
4	**Rectifying a Switchover Exception**	Rectify abnormal scenarios in the DR drill based on the post-check result.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Pre-Check Items

Check Item Description	Check Method	Possible Cause	Handling Procedure Used When the Results Are Not Passed
Run the **ping** command to test the connectivity of the data plane IP address. Run the **curl** command to test the connectivity of the port. Check the listening ports of the primary and standby clusters on the peer end.	Perform a two-way test on all nodes of the primary and standby clusters. The IP address is the data plane IP address. ping <ip> CN curl <ip>:8001 DN curl <ip>:30501 curl <ip>:30505	Before the switchover, the ports **baseport** plus 1 and plus 5 of the original primary cluster are in use, but the ports **baseport** plus 1 and plus 5 of the standby cluster are not in use. You need to ensure that the primary and standby clusters are properly connected after the switchover.	After confirming that the ports are enabled, perform the check again.
Check the cluster status (**Normal** or **Degraded**).	Run the **cm_ctl query -Cvp** command to query the value.	If one of the primary and standby clusters is in the unavailable state, the DR switchover fails.	Perform the query again after the cluster exception is rectified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Check Item Description	Check Method	Possible Cause	Handling Procedure Used When the Results Are Not Passed
Check whether the RPO is less than 10s and whether the RTO is less than or equal to 1s.	Call the OM Agent POST /v1/om/agent/mgr/cluster/streaming_disaster_recovery_query API to query the value. If the adaptor is connected, you can call the streaming_disaster_recovery_query API and directly observe the execution result.	If the RPO is too high, a large number of logs need to be passed to the standby cluster. If the RTO is too high, a large number of logs need to be replayed during the switchover.	Wait until the indicator value decreases to the required value.
Check whether the primary cluster is in the **archive** (incremental synchronizing) state and the standby cluster is in the **recovery** (incremental restoring) state.	Call the OM Agent POST /v1/om/agent/mgr/cluster/streaming_disaster_recovery_query API to query the value. If the adaptor is connected, you can call the streaming_disaster_recovery_query API and directly observe the execution result.	If the DR status is abnormal, the DR switchover fails.	Perform the query again after the DR exception is rectified.
Check the consistency point progress.	cm_ctl query -vg | grep Global_target_barrierId Global_target_barrierId: csn_00000000000017259643_170727842794 1 cm_ctl query -vg | grep Global_target_barrierId Global_target_barrierId: csn_00000000000017259645_170727842954 3	If the consistency point progress is abnormal, the log replay of the standby cluster is abnormal.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Performing a Switchover

If the prerequisites are met, perform the switchover using the following API:

Use the OM Agent URI /v1/om/agent/mgr/cluster/streaming_disaster_recovery_switchover to perform a switchover.

Performing a Post-check After a DR Drill

After the DR switchover is performed on the standby cluster for DR, call the OM Agent URI /v1/om/agent/mgr/cluster/dr_cluster_postcheck to check the cluster status, cluster type, and DR status, and report the IDs of the instances that are not involved in the DR switchover or the promotion to primary. Related information is used by the API caller to check the process execution result and determine the repair policy.

An example of the result returned by the API is as follows:

- **cluster_state**: cluster status.
- **hadr_cluster_stat**: DR status.
- **not_in_switchover_instance_ids**: IDs of the component instances that are not involved in the switchover.
- **incorrect_params_instance_ids**: IDs of the component instances whose parameters are incorrectly set.
- **hadr_primary_cluster_name**: cluster name of the primary cluster in the current DR system.

Rectifying a Switchover Exception

Handling basis: After the switchover process is complete, check the results returned by the post-check APIs of the two clusters. For details, see **Performing a Post-check After a DR Drill**.

- Scenario 1:

  In the **hadr_cluster_stat** returned by the post-check API, the DR status of the two clusters is not **archive**. In this case, there is no primary cluster after the switchover.

  Rectification solution:

  Call the POST /v1/om/agent/mgr/cluster/dr_cluster_manual_recovery API (API used to restore a cluster in the case of abnormal DR switchover) to restore the original primary cluster.

  If the DR relationship uses the cluster management mode, after the rectification is complete, call the POST /v1/om/agent/mgr/cluster/dr_cluster_serial_change API (the API used to update the DR sequence), and set **opt_type** to **set_dr_repl_dest_conf** to change the log-receiving cluster information.

- Scenario 2:

  In the **hadr_cluster_stat** returned by the post-check API, the DR status of the two clusters is not **recovery**. In this case, there is no standby cluster after the switchover.

  Rectification solution:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Call the POST /v1/om/agent/mgr/cluster/dr_cluster_manual_recovery API (API used to restore a cluster in the case of abnormal DR switchover) to switch the original primary cluster to the standby cluster for DR.

If the DR relationship uses the cluster management mode, after the rectification is complete, call the POST /v1/om/agent/mgr/cluster/dr_cluster_serial_change API (the API used to update the DR sequence), and set **opt_type** to **set_dr_repl_dest_conf** to change the log-receiving cluster information.

- Scenario 3:

  In the **hadr_cluster_stat** returned by the post-check API, the DR status of the original primary cluster is **archive**, and that of the original standby cluster is **recovery**. In this case, the switchover process is abnormal, both the primary and standby clusters are rolled back, and the DR relationship remains unchanged.

  Rectification solution:

  Rectify the abnormal component based on whether the following values returned by the post-check API are empty:

  - **not_in_switchover_instance_ids**: IDs of the component instances that are not involved in the switchover.
  - **incorrect_params_instance_ids**: IDs of the component instances whose parameters are incorrectly set. (If ['dr_dest_name', 'Incorrect'] exists, rectify **dr_dest_name** by referring to scenario 5.)

- Scenario 4:

  In the **hadr_cluster_stat** returned by the post-check API, the original primary cluster is switched to standby with the **recovery** state, and the original standby cluster is switched to primary with the **archive** state. In this case, the switchover process is abnormal, both the primary and standby clusters are not rolled back, the DR relationship is changed, and the switchover is complete.

  Rectify the abnormal component based on whether the following values returned by the post-check API are empty:

  - **not_in_switchover_instance_ids**: IDs of the component instances that are not involved in the switchover.
  - **incorrect_params_instance_ids**: IDs of the component instances whose parameters are incorrectly set. (If ['dr_dest_name', 'Incorrect'] exists, rectify **dr_dest_name** by referring to scenario 5.)

- Scenario 5:

  If **['dr_dest_name', 'Incorrect']** is present in the returned **incorrect_params_instance_ids**, call the POST /v1/om/agent/mgr/cluster/dr_cluster_serial_change API (the API used to update the DR sequence), and set **opt_type** to **set_dr_repl_dest_conf** to change the log-receiving cluster information.

?.1. Component Exceptions

After the IDs of the instances that are not involved in the DR switchover are reported using the post-check API, you can call the OM Agent URI /v1/om/agent/mgr/cluster/repair to restore the nodes.

Recovery scenarios

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Component Type	Fault Scenario
CMS(DCC)	The component is abnormal and does not participate in the DR switchover. As a result, the parameter is not set.
CMA	The component is abnormal and does not participate in the DR switchover. As a result, the parameter is not set.
ETCD	The component is abnormal and does not participate in the DR switchover. As a result, the parameter is not set.
CN	The component is abnormal and does not participate in the DR switchover. As a result, the status or parameters are abnormal.
DN	The component is abnormal and does not participate in the DR switchover. As a result, the status or parameters are abnormal.

?.2. Cluster Exceptions

During the switchover of the primary and standby clusters for DR, the switchover fails due to remote network exceptions or cluster component exceptions, you can call the OM Agent URI /v1/om/agent/mgr/cluster/dr_cluster_manual_recovery to restore the cluster to the primary cluster or standby cluster for DR.

- If there is no primary cluster (two standby clusters), call the kernel repair API to restore the original primary cluster to provide services.
- In the dual-primary scenario (two primary clusters), a new primary cluster provides services externally, and the original primary cluster is in the read-only state. Call the kernel repair API to restore the original primary cluster to the new standby cluster.

If the DR relationship uses the cluster management mode, after the rectification is complete, call the POST /v1/om/agent/mgr/cluster/dr_cluster_serial_change API (the API used to update the DR sequence), and set **opt_type** to **set_dr_repl_dest_conf** to change the log-receiving cluster information.

7.31.2.3.2 Intra-City Dual-DC HA Solution DR Drill Practice Reference

Introduction to DR Drills

The planned primary/standby cluster switchover function can be used to perform DR drill. This operation enables the roles of the primary and standby clusters to be exchanged. The original standby cluster becomes the primary cluster to provide services, and the original primary cluster becomes the standby cluster. This function ensures that no data is lost, that is, RPO is 0.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-56 Planned migration in the DR drill

Step	Operation	Description
1	**Pre-Check Items**	Check the prerequisites for DR switchover.
2	**Performing a Switchover**	Call the planned switchover API to perform a switchover for primary and standby clusters.
3	**Performing a Post-check After a DR Drill**	After the DR switchover is complete, check the cluster status, cluster type, and DR status, and report the IDs of instances that are not involved in the DR switchover. Use the related information to check the process execution result and determine the repair policy.
4	**Rectifying a Switchover Exception**	Rectify abnormal scenarios in the DR drill based on the post-check result.

Pre-Check Items

Check Item Description	Check Method	Possible Cause	Handling Procedure Used When the Results Are Not Passed
Run the **ping** command to test the connectivity of the data plane IP address. Run the **curl** command to test the connectivity of the port. Check the listening ports of the primary and standby clusters on the peer end.	Perform a two-way test on all nodes of the primary and standby clusters. The IP address is the data plane IP address. ping <ip> CN curl <ip>:39001 DN curl <ip>:45001 curl <ip>:45021. The port number of each shard is increased by 20.	Before the switchover, ports 39001 of the CN and 45001 of the DN in the original primary cluster are in use, but those in the standby cluster are not in use. You need to ensure that the primary and standby clusters are properly connected after the switchover.	After confirming that the ports are enabled, perform the check again.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Check Item Description	Check Method	Possible Cause	Handling Procedure Used When the Results Are Not Passed
Check the cluster status (**Normal**).	Run the **cm_ctl query -Cvp** command to query the value.	If one of the primary and standby clusters is not in the **Normal** or **Degraded** state, DR occurs.	Perform the query again after the cluster exception is rectified.
Check whether the RPO is 0s and whether the RTO is less than or equal to 1s.	Call the OM Agent POST /v1/om/agent/mgr/cluster/dorado_query API to query the value.	If the RPO is too high, a large number of logs need to be passed to the standby cluster. If the RTO is too high, a large number of logs need to be replayed during the switchover.	Wait until the indicator value decreases to the required value.
Check whether the primary cluster is in the **archive** (incremental synchronizing) state and the standby cluster is in the **recovery** (incremental restoring) state.	Call the OM Agent POST /v1/om/agent/mgr/cluster/dorado_query API to query the value.	If the DR status is abnormal, the DR switchover fails.	Perform the query again after the DR exception is rectified.
Check the consistency point progress.	cm_ctl query -vg | grep Global_target_barrierId Global_target_barrierId: csn_00000000000017259643_1707278427941 cm_ctl query -vg | grep Global_target_barrierId Global_target_barrierId: csn_00000000000017259645_1707278429543	If the consistency point progress is abnormal, the log replay of the standby cluster is abnormal.	-

Performing a Switchover

If the prerequisites are met, perform the switchover using the following API:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Use the OM Agent URI /v1/om/agent/mgr/cluster/dorado_switch to perform a planned switchover between the primary and standby clusters.

Performing a Post-check After a DR Drill

Use the OM Agent URI /v1/om/agent/mgr/cluster/dr_cluster_postcheck to perform DR drill post-check.

Perform the post-check in the backend as follows:

- For the primary cluster:

  curl -H "Content-Type:application/json" -d '{"drMode": " dorado_cluster ", "drOptionType": "switchover", "drClusterMode": "primary"}' -X POST $HOST_IP:port/v1/om/agent/mgr/cluster/dr_cluster_postcheck

- For the standby cluster:

  curl -H "Content-Type:application/json" -d '{"drMode": " dorado_cluster ", "drOptionType": "switchover", "drClusterMode": "disaster_standby"}' -X POST $HOST_IP:port/v1/om/agent/mgr/cluster/dr_cluster_postcheck

After the DR switchover process is complete, this API is used to check the cluster status, cluster type, and DR status, and report the IDs of the instances that are not involved in the DR switchover or the promotion to primary. Related information is used by the API caller to check the process execution result and determine the repair policy.

An example of the result returned by the API is as follows:

- **cluster_state**: cluster status.
- **hadr_cluster_stat**: DR status.
- **not_in_switchover_instance_ids**: IDs of the component instances that are not involved in the switchover.
- **incorrect_params_instance_ids**: IDs of the component instances whose parameters are incorrectly set.
- **hadr_primary_cluster_name**: cluster name of the primary cluster in the current DR system.

Rectifying a Switchover Exception

Handling basis: After the switchover process is complete, check the results returned by the post-check APIs of the two clusters. For details, see **Performing a Post-check After a DR Drill**.

- Scenario 1: **hadr_cluster_stat** returned by the post-check API indicates that the two clusters are in the **recovery** state. In this case, there is no primary cluster after the switchover.

  Solution: Rectify the fault by referring to **Cluster Exceptions**.

- Scenario 2: **hadr_cluster_stat** returned by the post-check API indicates that the original primary cluster is in the **archive** state and the original standby cluster is in the **recovery** state. In this case, the switchover process is abnormal, the original primary cluster is rolled back, and the DR relationship remains unchanged.

  Solution: Rectify the fault by referring to **Component Exceptions**.

- Case 3: **hadr_cluster_stat** returned by the post-check API indicates that the original primary cluster is switched to standby with the **recovery** state and

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the original standby cluster is switched to primary with the **archive** state. In this case, the switchover process is abnormal, but the DR relationship is changed and the switchover is complete.

Solution: Rectify the fault by referring to **Component Exceptions**.

?.1. Component Exceptions

Rectify the abnormal component based on whether the following values returned by the post-check API (see "Performing a Post-check After a DR Drill") are empty:

- **not_in_switchover_instance_ids**: IDs of the component instances that are not involved in the switchover.
- **incorrect_params_instance_ids**: IDs of the component instances whose parameters are incorrectly set.

API for recovering a node:

POST /v1/om/agent/mgr/cluster/repair

Recovery scenarios

Component Type	Fault Scenario
CMS(DCC)	The component is abnormal and does not participate in the DR switchover. As a result, the parameter is not set.
CMA	The component is abnormal and does not participate in the DR switchover. As a result, the parameter is not set.
CN	The component is abnormal and does not participate in the DR switchover. As a result, the status or parameters are abnormal.
DN	The component is abnormal and does not participate in the DR switchover. As a result, the status or parameters are abnormal.

?.2. Cluster Exceptions

If the switchover fails due to remote network exceptions or cluster component exceptions, you can use this API to restore the cluster to the primary cluster or standby cluster for DR. You can use the API for restoring clusters in abnormal DR switchover scenarios to rectify the fault.

- API for restoring a cluster:

  POST /v1/om/agent/mgr/cluster/dr_cluster_manual_recovery
- Query phase API:

  If an adaptor is connected, you can use dorado_switchover_precheck to check whether the primary cluster is switched to the standby cluster. After the primary cluster is switched to the standby cluster, the standby cluster checks that the primary cluster is successfully switched to the standby cluster (that is, the flag **Reduce standby success** exists) to provide a basis for the standby cluster to be switched to the primary cluster. If the flag is not written, you cannot ensure that all logs have been synchronized. Therefore, you need to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

restore the original primary cluster. If the flag has been written, all logs have been synchronized between the primary and standby clusters, and you can restore the original standby (new primary) cluster.

Scenario 1: If this flag is not written, the primary-to-standby process is automatically rolled back and the primary cluster is automatically restored. If the rollback is abnormal, there are two standby clusters for DR without a primary instance. In this case, use the kernel repair API to rectify the fault and quickly restore the original primary cluster to provide services.

Scenario 2: If the flag has been written, the standby-to-primary process has been started. In the abnormal standby-to-primary process scenario where there are two standby clusters for DR without a primary cluster, use the kernel repair API to quickly restore the original standby (new primary) cluster to provide services.

Example of backend recovery command: **curl -H "Content-Type:application/json" -d '{"drMode": "dorado_cluster", "drClusterMode": "primary"}' -X POST** *$HOST_IP*:*$PORT*/**v1/om/agent/mgr/cluster/dr_cluster_manual_recovery**

- *$HOST_IP*: management IP address of the node where the command is executed.
- *$PORT*: OM Agent listening port.

7.31.2.4 Typical Issues

1. For details about the streaming replication-based remote DR solution, see **Exceptions in Planned Switchover for Two-City Three-DC DR**.
2. For details about the intra-city dual-DC HA solution, see **Exceptions in Planned Switchover of Intra-City Dual-DC**.

7.31.2.4.1 Exceptions in Planned Switchover for Two-City Three-DC DR

Symptom

- **Symptom 1:**

  The following error is returned during a planned switchover on the primary cluster:

  ```
  Result exception error : Failed to generate switchover barrier before switchover
  ```

- **Symptom 2:**

  The following error is returned during a planned switchover on the primary cluster:

  ```
  Result exception error : Failed to truncate service before switchover
  ```

- **Symptom 3:**

  The following error is returned during a planned switchover on the standby cluster for DR:

  ```
  Result exception error : Failed to do check switchover_barrier on all main standby dn and cn. Because check timeout: XXs
  ```

- **Symptom 4:**

  After the quick switchover mode is enabled for a planned switchover, a switchover fails.

- **Symptom 5:**

  A faulty node does not participate in a switchover for DR.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Symptom 6:

A planned switchover is successful, but the process of the original primary DN or CN in the original primary cluster exits during the switchover.

Cause Analysis

- **Cause for symptom 1:**

  When the primary cluster receives a planned switchover command, a switchover barrier is generated before the primary cluster is demoted to standby. This barrier ensures that the replication of logs of all CNs and DN shards in the primary and standby clusters stops at a specified checkpoint. If a switchover barrier fails to be generated on the primary cluster due to reasons such as network jitter on the primary cluster, the planned switchover will be canceled.

- **Cause for symptom 2:**

  When the primary cluster receives a planned switchover command, a switchover barrier is generated before the primary cluster is demoted to the standby cluster for DR. In addition, services are truncated to terminate log writing. This barrier ensures that the replication of logs of all CNs and DN shards in the primary and standby clusters stops at a specified checkpoint. If services on the primary cluster fail to be truncated because the CN/DN instance on the primary cluster is abnormal, the remote network is abnormal, or the standby cluster for DR does not respond, the planned switchover is canceled.

- **Cause for symptom 3:**

  After the standby cluster receives a planned switchover command, it checks whether a switchover barrier has been received on all CNs and the main standby DN before being promoted to primary. This barrier ensures that the replication of logs of all CNs and DN shards in the primary and standby clusters stops at a specified checkpoint. If the standby cluster fails to obtain the switchover barrier within the specified timeout interval due to reasons such as abnormal remote network, the planned switchover will be canceled.

- **Cause for symptom 4:**

  There are many causes for quick switchover failures. Simple fault locating methods to quickly locate and demarcate faults are as follows:

  a. Query the latest OM log (*$GAUSSLOG*/**om/gs_om***) by keyword to determine whether the switchover fails and rollback exists.

  b. Locate the step where the quick switchover fault occurs based on the details in the queried error information.

  c. Based on the failure cause located in the previous step, view the logs of OM Agent or gs_log on the corresponding cluster node to obtain the specific failure cause.

  A fault locating case is provided to describe the preceding process.

  If the error log "Failed to do streaming disaster cluster switchover" is displayed and the error information is "switchover timeout", search for "switchover" in gs_log on the primary and main standby nodes based on the printed instances to determine where the kernel switchover fails or times out. Identify the cause based on the printed information about sent and received messages, for example, the network interaction is abnormal, logs are not all sent within a specified period, or the thread exit times out.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Cause for symptom 5:**

  The node does not participate in the switchover due to server breakdown or network interruption.

  Cause for symptom 6:

  Search for "abort switchover" in the process logs. If the keyword exists, the process exits because the demotion times out. This is normal. Wait until the process is restarted by the CM. If the keyword does not exist, further check the logs to determine the cause.

Solution

Step 1 If the **switchover** command executed on the primary cluster to demote it to standby or on the standby cluster for DR to promote it to primary times out or fails, the cluster rolls back to the status before the switchover. The planned switchover can be performed again on the primary and standby clusters.

Step 2 If multiple switchover operations fail, you need to further analyze the log files related to streaming DR.

----End

The troubleshooting procedure for quick switchover is as follows:

Step 1 Use the cm_ctl tool to check whether the clusters are switched over or rolled back and whether the instances are normal. For faulty nodes, you can use the node recovery functionality.

```
cm_ctl query -Cvp
```

Example of a primary cluster in the **Normal** state:

```
[  CMServer State   ]

node            instance state
-------------------------------------
1  <IP>  1       Primary
2  <IP>  2       Standby
3  <IP>  3       Standby

[    ETCD State     ]

node            instance state
-----------------------------------------
1  <IP>  7001     StateFollower
2  <IP>  7002     StateLeader
3  <IP>  7003     StateFollower

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[ Coordinator State ]

node            instance       state
-------------------------------------
1  <IP>  5001 <PORT>      Normal
2  <IP>  5002 <PORT>      Normal
3  <IP>  5003 <PORT>      Normal

[ Central Coordinator State ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
node            instance state
-----------------------------------
1  <IP>  5001     Normal

[    GTM State    ]

node            instance state                sync_state
-------------------------------------------------------------------
1  <IP>  1001     P Primary Connection ok  Sync
2  <IP>  1002     S Standby Connection ok  Sync
3  <IP>  1003     S Standby Connection ok  Sync

[  Datanode State   ]

node            instance       state          | node            instance       state          | node            instance
state
-----------------------------------------------------------------------------------------------------------------------------
---------------------------------
1  <IP>  6001 <PORT>       P Primary Normal | 2  <IP>  6002 <PORT>       S Standby Normal | 3  <IP>  6003
<PORT>       S Standby Normal
2  <IP>  6004 <PORT>       P Primary Normal | 3  <IP>  6005 <PORT>       S Standby Normal | 1  <IP>  6006
<PORT>       S Standby Normal
3  <IP>  6007 <PORT>       P Primary Normal | 1  <IP>  6008 <PORT>       S Standby Normal | 2  <IP>  6009
<PORT>       S Standby Normal
```

Example of a standby cluster for DR in the **Normal** state:

```
[  CMServer State   ]

node            instance state
-----------------------------------
1  <IP>  1        Primary
2  <IP>  2        Standby
3  <IP>  3        Standby

[    ETCD State    ]

node            instance state
-----------------------------------------
1  <IP>  7001     StateLeader
2  <IP>  7002     StateFollower
3  <IP>  7003     StateFollower

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[ Coordinator State ]

node            instance       state
-----------------------------------
1  <IP>  5001 <PORT>       Normal
2  <IP>  5002 <PORT>       Normal
3  <IP>  5003 <PORT>       Normal


[    GTM State    ]

node            instance state                sync_state
-------------------------------------------------------------------
1  <IP>  1001     P Primary Connection ok  Sync
2  <IP>  1002     S Standby Connection ok  Sync
3  <IP>  1003     S Standby Connection ok  Sync

[  Datanode State   ]

node            instance       state          | node            instance       state          | node            instance
state
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------
1  <IP>  6001 <PORT>      P Main Standby Normal | 2  <IP>  6002 <PORT>      S Cascade Standby Normal |
3  <IP>  6003 <PORT>      S Cascade Standby Normal
2  <IP>  6004 <PORT>      P Main Standby Normal | 3  <IP>  6005 <PORT>      S Cascade Standby Normal |
1  <IP>  6006 <PORT>      S Cascade Standby Normal
3  <IP>  6007 <PORT>      P Main Standby Normal | 1  <IP>  6008 <PORT>      S Cascade Standby Normal |
2  <IP>  6009 <PORT>      S Cascade Standby Normal
```

Step 2 Use the gs_om tool to query the parameters of the cluster instance that fails to be switched over.

```
echo "password" | gs_om -t dr_post_check -c [switchover|failover|query] -m [primary|disaster_standby] --
dr_type streaming_cluster
```

- Parameters related to the primary database instance:
  - CM Server parameters: **backup_open=0**, **dn_arbitrate_mode=quorum**, **enable_transaction_read_only=on**, **cm_hadr_process_type=none**.
  - CM Agent parameters: **agent_backup_open=0**, **enable_auto_start=1**, **cm_hadr_process_type=none**.
  - ETCD parameter: **/xxxx/CMServer/backup_open=0**.
  - CN parameters: **stream_cluster_run_mode=cluster_primary**, **hadr_process_type=none**, **auto_csn_barrier=1**, and **default_transaction_read_only=off**.
  - DN parameters: **stream_cluster_run_mode=cluster_primary**, **hadr_process_type=none**, and **default_transaction_read_only=off**.
- Parameters related to the standby database instance for DR:
  - CM Server parameters: **backup_open=2**, **dn_arbitrate_mode=quorum**, **enable_transaction_read_only=on**, **cm_hadr_process_type=none**.
  - CM Agent parameters: **agent_backup_open=2**, **enable_auto_start=1**, **cm_hadr_process_type=none**.
  - ETCD parameter: **/xxxx/CMServer/backup_open=2**.
  - CN parameters: **stream_cluster_run_mode=cluster_standby**, **hadr_process_type=none**, **auto_csn_barrier=0**, and **default_transaction_read_only=off**.
  - DN parameters: **stream_cluster_run_mode=cluster_standby**, **hadr_process_type=none**, and **default_transaction_read_only=off**.

Step 3 In extreme scenarios where there are dual standby cluster instances for DR, manual intervention is required.

- Dual standby database instances for DR indicate that the primary cluster is demoted to standby and is not rolled back, and the standby cluster is rolled back during promotion. In this case, data cannot be written to the two clusters.
- Use the gs_om tool to repair the original primary database instance and restore it to primary.

  ```
  echo "password" | gs_om -t dr_cluster_manual_recovery -m primary --dr_type streaming_cluster [--
  time-out=SECS]
  ```

- After the restoration is successful, disable the read-only mode and set the **default_transaction_read_only** parameter to **off**.

Step 4 In extreme scenarios where there are dual primary cluster instances, manual intervention is required.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Dual primary clusters indicate that the standby cluster for DR is successfully promoted to primary with the read-only mode disabled and the primary cluster fails to be demoted to standby and is rolled back with the read-only mode enabled.
- Use the gs_om tool to repair the original primary cluster and change it to the standby cluster.
  ```
  echo "password" |gs_om -t dr_cluster_manual_recovery -m disaster_standby --dr_type streaming_cluster [--time-out=SECS]
  ```
- Because the original primary cluster is read-only, no customer data can be imported. However, logs are still forked, and the restoration takes time. After the restoration is successful, disable the read-only mode and set the **default_transaction_read_only** parameter to **off**.

  NOTE

  - When the restoration tool (dr_cluster_manual_recovery) described in **Step 3** and **Step 4** is used to restore a cluster, if logs are stacked in the cluster, the restoration duration is affected by the number of logs to be replayed after the cluster is restarted.
  - You can check the gs_log file of the corresponding cluster node. If "xlog redo at lsn X/XXX" is continuously printed in the log and the node is in the **STARTING** state, the node is replaying stacked logs. Wait until the restoration process is complete.

Step 5 If the DR relationship uses the cluster management mode and **['dr_dest_name', 'Incorrect']** is present in the returned **incorrect_params_instance_ids**, call the POST /v1/om/agent/mgr/cluster/dr_cluster_serial_change API (the API used to update the DR sequence), and set **opt_type** to **set_dr_repl_dest_conf** to change the log-receiving cluster information.

Step 6 After operations on the faulty node such as hardware repair and power-on, add the node back to the cluster.

NOTICE

After the node is added to the cluster, the cluster and the instance on the node may both be in the **Normal** state. However, some parameters of the instance on the node are incorrectly configured, so you need to perform the following operations.

- Use the post-check API to query the instances that are not involved in the switchover for DR.
  ```
  echo "password" | gs_om -t dr_post_check -c switchover -m primary|disaster_standby --dr_type streaming_cluster
  ```
- Run the **cm_ctl stop** command to stop the CNs, DNs, and CM Servers that are not involved in the DR switchover.
  ```
  cm_ctl stop -n NODEID -D DATADIR
  ```

Step 7 Delete the residual archive slots.

- Check whether there are residual archive slots on the CNs and DNs.
  ```
  SELECT slot_name FROM pg_catalog.pg_get_replication_slots() where slot_type='physical' and slot_name not in ('gs_roach_full', 'gs_roach_inc', 'gs_roach_table_full', 'gs_roach_table_inc') and slot_name not like 'cn_%' and slot_name not like 'dn_%';
  ```
- If an archive slot exists, run the following SQL statement to delete it (**slot_name** indicates the queried archive slot name):
  ```
  SELECT * FROM pg_catalog.pg_drop_replication_slot('slot_name');
  ```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.31.2.4.2 Exceptions in Planned Switchover of Intra-City Dual-DC

Symptom

- Symptom 1:

  The primary-to-standby demotion fails. Logs cannot be viewed due to **primary_demote** timeout.

- Symptom 2:

  During the standby-to-primary promotion, the waiting for the cluster status to restore to **normal** times out. The cluster status is **Unavailable**. Some shards have no primary DN.

- Symptom 3:

  A switchover for DR fails.

- Symptom 4:

  A faulty node does not participate in a switchover for DR.

- Symptom 5:

  A planned switchover is successful, but the process of the primary DN or CN in the primary cluster exits during the primary-to-standby demotion.

Cause Analysis

- Cause for symptom 1:

  The logs of the failed node show that a large number of dirty pages need to be refreshed at the checkpoint, for which the time required exceeds the timeout interval of **primary_demote**.

- Cause for symptom 2:

  The time required for log replay exceeds the timeout interval of the promotion in the switchover process (300 seconds). As a result, the cluster cannot be restored to the **normal** or **degraded** state within 300 seconds, and the promotion fails.

- Cause for symptom 3:

  There are many causes for switchover failures. Simple fault locating methods to quickly locate and demarcate faults are as follows:

  a. Query the latest OM log (*$GAUSSLOG*/**om/gs_om***) by keyword to determine whether the switchover fails and rollback exists.

  b. Locate the fault based on the details in the queried error information or the procedure file **do_dorado_switch.dat** in the *$PGHOST* directory.

  c. Based on the failure cause located in the previous step, view the logs of OM Agent or gs_log on the corresponding cluster node to obtain the specific failure cause.

- Cause for symptom 4:

  The node does not participate in the switchover due to server breakdown or network interruption.

- Cause for symptom 5:

  Search for "abort switchover" in the process logs. If the keyword exists, the process exits because the demotion times out. This is normal. Wait until the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

process is restarted by the CM. If the keyword does not exist, further check the logs to determine the cause.

Solution

- Solution to symptom 1:

Step 1 Wait until the primary DN is successfully demoted to standby.

Step 2 Reenter the primary-to-standby switchover process.

----End

- Solution to symptom 2:

Step 1 Wait until the cluster is restored to the **normal** or **degraded** state.

Step 2 Run the **switchover** command again.

----End

The procedure for troubleshooting the switchover process is as follows:

Step 1 Use the cm_ctl tool to check whether the clusters are switched over or rolled back and whether the clusters are normal. For faulty nodes, you can use the node recovery functionality.

```
cm_ctl query -Cvp
```

Example of a primary cluster in the **Normal** state:

```
[  CMServer State   ]

node             instance state
-----------------------------------
1  <IP>  1       Primary
2  <IP>  2       Standby
3  <IP>  3       Standby

[    ETCD State     ]

node             instance state
-----------------------------------------
1  <IP>  7001     StateFollower
2  <IP>  7002     StateLeader
3  <IP>  7003     StateFollower

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[ Coordinator State ]

node             instance       state
-----------------------------------
1  <IP>  5001 <PORT>      Normal
2  <IP>  5002 <PORT>      Normal
3  <IP>  5003 <PORT>      Normal

[ Central Coordinator State ]

node             instance state
-----------------------------------
1  <IP>  5001     Normal
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
[    GTM State    ]

node             instance state                 sync_state
-----------------------------------------------------------------
1  <IP>  1001     P Primary Connection ok  Sync
2  <IP>  1002     S Standby Connection ok  Sync
3  <IP>  1003     S Standby Connection ok  Sync

[  Datanode State   ]

node             instance       state          | node             instance       state          | node             instance
state
-----------------------------------------------------------------------------------------------------------------------------
---------------------------------
1  <IP>  6001 <PORT>       P Primary Normal | 2  <IP>  6002 <PORT>       S Standby Normal | 3  <IP>  6003
<PORT>       S Standby Normal
2  <IP>  6004 <PORT>       P Primary Normal | 3  <IP>  6005 <PORT>       S Standby Normal | 1  <IP>  6006
<PORT>       S Standby Normal
3  <IP>  6007 <PORT>       P Primary Normal | 1  <IP>  6008 <PORT>       S Standby Normal | 2  <IP>  6009
<PORT>       S Standby Normal
```

Example of a standby cluster for DR in the **Normal** state:

```
[  CMServer State   ]

node             instance state
---------------------------------
1  <IP>  1        Primary
2  <IP>  2        Standby
3  <IP>  3        Standby

[    ETCD State     ]

node             instance state
-----------------------------------------
1  <IP>  7001     StateFollower
2  <IP>  7002     StateLeader
3  <IP>  7003     StateFollower

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[ Coordinator State ]

node             instance       state
---------------------------------
1  <IP>  5001 <PORT>       Normal
2  <IP>  5002 <PORT>       Normal
3  <IP>  5003 <PORT>       Normal

[ Central Coordinator State ]

node             instance state
---------------------------------
1  <IP>  5001     Normal

[    GTM State    ]

node             instance state                 sync_state
-----------------------------------------------------------------
1  <IP>  1001     P Primary Connection ok  Sync
2  <IP>  1002     S Standby Connection ok  Sync
3  <IP>  1003     S Standby Connection ok  Sync

[  Datanode State   ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
node          instance       state          | node          instance       state          | node          instance
state
-----------------------------------------------------------------------------------------------------------------------------------
---------------------------------
1  <IP>  6001 <PORT>      P Standby Normal | 2  <IP>  6002 <PORT>      S Standby Normal | 3  <IP>  6003
<PORT>      S Standby Normal
2  <IP>  6004 <PORT>      P Standby Normal | 3  <IP>  6005 <PORT>      S Standby Normal | 1  <IP>  6006
<PORT>      S Standby Normal
3  <IP>  6007 <PORT>      P Standby Normal | 1  <IP>  6008 <PORT>      S Standby Normal | 2  <IP>  6009
<PORT>      S Standby Normal
```

Step 2 Use the gs_om tool to query the parameters of the cluster that fails to be switched over.

```
gs_om -t dr_post_check -c [switchover|failover|query|restore] -m [primary|disaster_standby] --dr_type
dorado_cluster
```

- Parameters related to the primary cluster:
  - CM Server parameters: **backup_open=0**, **dn_arbitrate_mode=quorum**, **cm_hadr_process_type=none**.
  - CM Agent parameters: **agent_backup_open=0**, **enable_auto_start=1**, **cm_hadr_process_type=none**.
  - ETCD parameter: **/xxxx/CMServer/backup_open=0**.
  - CN parameters: **cluster_run_mode=cluster_primary**, **hadr_process_type=none**, **auto_csn_barrier=1**, and **default_transaction_read_only=off**.
  - DN parameters: **cluster_run_mode=cluster_primary**, **hadr_process_type=none**, and **default_transaction_read_only=off**.
- Parameters related to the standby cluster for DR:
  - CM Server parameters: **backup_open=1**, **dn_arbitrate_mode=quorum**, **cm_hadr_process_type=none**.
  - CM Agent parameters: **agent_backup_open=1**, **enable_auto_start=1**, **cm_hadr_process_type=none**.
  - ETCD parameter: **/xxxx/CMServer/backup_open=1**.
  - CN parameters: **cluster_run_mode=cluster_standby**, **hadr_process_type=none**, **auto_csn_barrier=0**, and **default_transaction_read_only=off**.
  - DN parameters: **cluster_run_mode=cluster_standby**, **hadr_process_type=none**, and **default_transaction_read_only=off**.

Step 3 If an instance that is not involved in the switchover or whose parameters are incorrectly configured is detected by the post-check API, the extreme scenario where there are dual standby clusters for DR occurs and manual intervention is required.

- No primary cluster is available in the cluster in either of the following scenarios:

  1. Before "Reduce standby success" is written, the primary cluster fails to be demoted to standby. The rollback is started and then fails.

  2. The standby cluster for DR fails to be promoted to primary.

  Use the gs_om tool to quickly restore the primary cluster to provide services externally.

  ```
  echo "password" | gs_om -t dr_cluster_manual_recovery -m primary --dr_type dorado_cluster [--time-
  out=SECS]
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 4 After operations on the faulty node such as hardware repair and power-on, add the node back to the cluster.

NOTICE

After the node is added to the cluster, the cluster and the cluster on the node may both be in the **Normal** state. However, some parameters of the cluster on the node are incorrectly configured, so you need to perform the following operations.

- Use the post-check API to query the clusters that are not involved in the switchover for DR.
  ```
  gs_om -t dr_post_check -c switchover -m primary|disaster_standby --dr_type dorado_cluster
  ```
- Run the **cm_ctl stop** command to stop the CNs, DNs, and CM Servers that are not involved in the DR switchover.
  ```
  cm_ctl stop -n NODEID -D DATADIR
  ```

----End

7.32 Rebuilding Primary and Standby Nodes

7.32.1 Overview

Primary/standby rebuild is a core feature in the **HA system** of the GaussDB services. It **performs full or incremental data synchronization from the primary node or a complete standby node** to resolve data inconsistency between the primary and standby nodes due to data latency, disk damage, log breakage, or other faults on the standby node. This enables quick takeover by the standby node and restoration of cluster availability. This feature is a key technical guarantee for achieving cluster HA and ensuring eventual data consistency.

Description

The **WAL-based physical replication and replay** technology is used to solve **data inconsistency** caused by standby node faults, breakdowns, or data damage, and to **automatically or manually perform full or incremental data rebuild on the standby node**. This ensures eventual data consistency between the primary and standby clusters and guarantees the continuous HA of the cluster.

Scenarios

- **Industries**: industries that have high requirements on data reliability and service continuity, such as finance (payment and settlement), telecommunications (charging and accounting), and government (unified online government services).
- **Pain points:**
  - **Hardware faults**: The disk of the server where the standby node is located is damaged. As a result, data files are lost or unavailable.
  - **Network faults**: The standby node is disconnected from the primary node for a long time, and the stacked logs have been cleared by the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

primary node. As a result, the log breakpoint occurs and synchronization cannot be performed through common log replay.

- **Manual misoperations**: Data is polluted due to incorrect write operations performed on the standby node.

- **Common trigger conditions**:
  - Automatic HA recovery in a cluster
    i. **WAL segment removed**: The LSN requested by the standby node does not exist on the primary node. As a result, the log replication between the primary and standby nodes is interrupted.
    ii. **System id not match**: The **system identifier** recorded on the standby node is inconsistent with that on the primary node. This occurs when a node is added to the cluster, for example, during cluster installation, node replacement, or copy addition. In this case, data needs to be synchronized through **full rebuild**.
    iii. **DCF log loss**: According to the DCF-based consistency replication protocol, when the standby node detects that the maximum **DCF index** recorded on the standby node does not exist on the primary node, the standby node needs to be rebuilt for recovery.
  - Manual triggering

    In some special scenarios, rebuild needs to be manually triggered for recovery.

    For example, during peak hours, the network resources and compute resources of the primary node are limited and automatic recovery cannot be triggered. In this case, rebuild needs to be performed during off-peak hours.
  - Triggering during the cluster management process
    i. Establishment and resumption of a standby cluster for DR
    ii. Node recovery and replacement
    iii. Backup and restoration

Specifications and Limitations

- **Compatibility**:
  - **Supported versions**: (The actual version numbers prevail.)

    Table 7-57 Versions that support primary/standby rebuild

Feature	Version
Full/Incremental rebuild of primary/standby nodes	GaussDB V500R001C00 or later
Incremental rebuild from standby to standby	GaussDB 505.1.0 or later
Asynchronous build	GaussDB 505.2.1 spc0800 and GaussDB 507.0 or later

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Feature	Version
Parallel rebuild and parallel compression and rebuild	GaussDB 506.0.0 or later

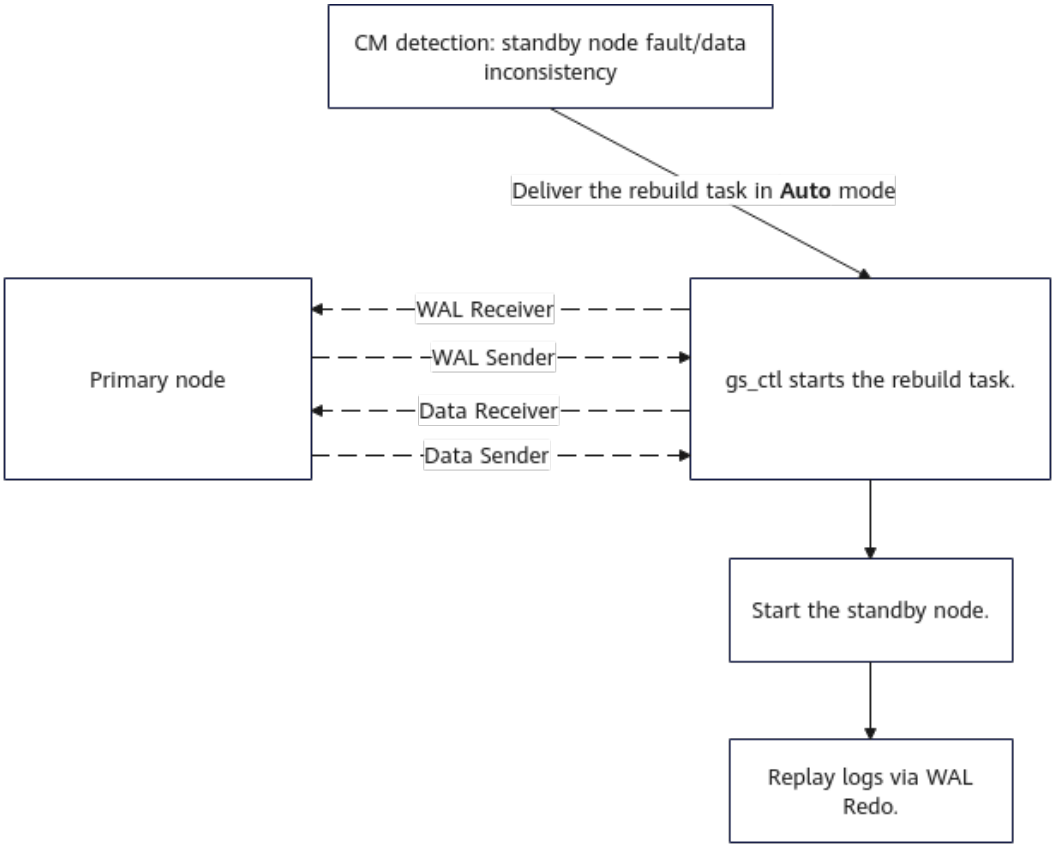
  - **Ecosystem tools**: OM (O&M management service), CM (database management service), and gs_ctl (command line tool).
- **Performance limitations**:
  - **Network bandwidth**: The rebuild process involves data transmission, which consumes network bandwidth. Therefore, you are advised to avoid peak hours.
  - **Primary node load**: During the rebuild, the primary node needs to read data pages and send data, which affects the performance of the primary node (increasing the CPU and I/O pressure).
  - **Disk I/O**: The target standby node needs to receive and write a large amount of data. If the network is not restricted, the disk I/O is the main bottleneck.
- **Functionality restrictions**:
  - **Uniqueness of the primary node**: The CM allows the primary node to provide the rebuilding service for only one standby node at a time (but the primary node can rebuild data for multiple standby nodes at the same time).
  - **Mutual exclusion of the rebuild process**: A standby node cannot be rebuilt from two different source nodes at the same time.
  - **Cascaded standby node**: The source node for rebuilding a cascaded standby node can only be its direct upstream node.

Technical Architecture

The following figure shows the core architecture of GaussDB primary/standby rebuild, involving core components such as **CM, WAL Sender, WAL Receiver, Data Sender, Data Receiver**, and **WAL Redo**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-65 Rebuild process



Core components:

- **CM**: cluster HA management service. It is responsible for detecting the node status, determining whether to trigger rebuild when the standby node is abnormal, and coordinating the start and stop of the rebuild task.
- **WAL Sender**: WAL sending thread on the primary node. It is responsible for sending WAL data generated during rebuild to the standby node. At the beginning and end of a rebuild task, a consistency snapshot point (checkpoint) is sent to the standby node as the start and end points of the replay.
- **Data Sender**: data sending thread on the primary node. It is responsible for reading complete data files from disks and streaming them to the standby node.
- **WAL Receiver**: WAL receiving process on the standby node. It is responsible for receiving WAL data sent from the primary node.
- **Data Receiver**: data receiving thread on the standby node. It is responsible for receiving data files from the primary node and writing them to the local disk.
- **WAL Redo**: WAL replaying thread on the standby node. After receiving all data and starting the standby node, the thread replays all WALs generated after the rebuild start point to ensure that the data on the standby node is the same as that on the primary node.

Data flow direction:

1. **Data link**: All data files are transmitted from the **Data Sender** on the primary node to the **Data Receiver** on the standby node and flushed to disks.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. **Log link**: WALs are transmitted from the **WAL Sender** on the primary node to the **WAL Receiver** on the standby node and flushed to disks. After the standby node is started, the **WAL Redo** thread replays the WALs.

Decision-making points:

- **Rebuild mode selection**: Except in special scenarios, the CM service delivers the **Auto Build** mode by default, which means that an incremental rebuild is performed first. If the incremental rebuilding fails for three times, a full rebuild is performed.

Core components:

- **CM**: cluster HA management service. It is responsible for detecting the node status, determining whether to trigger rebuild when the standby node is abnormal, and coordinating the start and stop of the rebuild task.
- **WAL Sender**: WAL sending thread on the primary node. It is responsible for sending WAL data generated during rebuild to the standby node. At the beginning and end of a rebuild task, a consistency snapshot point (checkpoint) is sent to the standby node as the start and end points of the replay.
- **Data Sender**: data sending thread on the primary node. It is responsible for reading complete data files from disks and streaming them to the standby node.
- **WAL Receiver**: WAL receiving process on the standby node. It is responsible for receiving WAL data sent from the primary node.
- **Data Receiver**: data receiving thread on the standby node. It is responsible for receiving data files from the primary node and writing them to the local disk.
- **WAL Redo**: WAL replaying thread on the standby node. After receiving all data and starting the standby node, the thread replays all WALs generated after the rebuild start point to ensure that the data on the standby node is the same as that on the primary node.

Data flow direction:

1. **Data link**: All data files are transmitted from the **Data Sender** on the primary node to the **Data Receiver** on the standby node and flushed to disks.
2. **Log link**: WALs are transmitted from the **WAL Sender** on the primary node to the **WAL Receiver** on the standby node and flushed to disks. After the standby node is started, the **WAL Redo** thread replays the WALs.

Decision-making points:

- **Rebuild mode selection**: Except in special scenarios, the CM service delivers the **Auto Build** mode by default, which means that an incremental rebuild is performed first. If the incremental rebuilding fails for three times, a full rebuild is performed.

7.32.2 Technical Principle

Core Mechanism

1. **Consistent snapshot synchronization (Establish Snapshot)**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

When the rebuild starts, the WAL Sender thread on the primary node obtains a current consistency point (checkpoint) and sends the log sequence number (LSN) and timeline information of the point to the standby node. This snapshot point ensures that the start point of the rebuilt data is consistent.

2. **Streaming transmission of full data files (Streaming Base Backup)**

   The Data Sender thread on the primary node reads all data files based on the previously obtained consistent snapshot point, and sends the data files to the Data Receiver thread on the standby node in the form of data streams through TCP connections. The Data Receiver thread writes the data files to the corresponding locations in the local data directory.

3. **Incremental log replay (WAL Redo)**

   After the data transmission starts, the WAL Sender thread on the primary node continuously sends the WALs generated after the snapshot point to the standby node. The standby node receives and stores these logs. After all data is received, the WAL Redo thread replays these logs in sequence from the snapshot point to catch up with the latest data.

Feature Description

1. **Full Build**
   - Description

     The data files of the entire primary node are **physically replicated** to provide a new and complete data copy for the faulty standby node. During the data synchronization, all WALs generated on the primary node are replayed to ensure data consistency.

     This feature is characterized by a large amount of data to be replicated and long time required.
   - Application scenarios
     i. The disk is damaged, or data files are lost or damaged.
     ii. The WALs on the standby node are inconsistent from those on the primary node (the WALs required for replay have been cleared by the primary node).
     iii. Some cluster management processes exist, such as adding a node to the cluster.
     iv. Data is polluted due to misoperations.
2. **Incremental Build**
   - Description

     After log forking (WAL segment removed) occurs on the primary and standby nodes, the LSN corresponding to the latest forking point is found by comparing the local WALs on the primary and standby nodes. Data files are synchronized from the primary node, overwriting the files on the standby node after the forking point. The WALs after the forking point on the primary node are also synchronized and replayed to ensure data consistency.

     Only data changes during the fault period are synchronized, instead of the complete data set. This mode is more efficient than full build.

     Currently, this mode is the default recovery mode for intra-cluster HA.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Application scenarios
  i. The standby node breaks down due to a process exception, but the data directory is intact. Only a small amount of data is required for data restoration.
  ii. During an intra cluster failover, the original primary node is demoted to standby, and log forking occurs.
  iii. A standby cluster for DR is removed and resumed.

3. **Parallel Build**
   - Description

     Currently, full and incremental rebuilds in GaussDB are implemented through **single-thread** data transmission. In large-capacity scenarios, rebuilds are too slow to meet customer requirements. To address the issue, **parallel build** introduces the **multi-thread** parallel data transmission mechanism based on full and incremental rebuilds. This fully utilizes hardware resources and improves the rebuild speed.

     - Parallel rebuild
       - Data is transmitted in parallel using multiple threads, with adjustable thread count for optimal utilization of network bandwidth and storage device I/O. You can set the GUC parameter **parallel_build_thread_num** to control the degree of parallelism for parallel build.
     - Compression and rebuild
       - The LZ4 lossless compression algorithm is used to compress the data to be transmitted, saving network bandwidth. You can set the GUC parameter **enable_parallel_build_compression** to determine whether to compress data during parallel rebuild.
   - Application scenarios

     For data nodes with data volume at the TB level or above, full rebuild is required for restoration.
   - Limitations

     This feature can be enabled to improve the rebuild efficiency when the CPU, storage device I/O, and network bandwidth are sufficient. Otherwise, no obvious gain can be observed due to limited hardware resources.

4. **Async Build**
   - Description

     The core purpose of async build is to **ensure the continuity and availability of production services on the primary node** when a large number of nodes are faulty. This prevents the primary node from being suspended or blocked due to insufficient standby nodes, and allows the faulty standby nodes to quickly return to the cluster in a degraded state to participate in arbitration.

     When continuous region-level faults are triggered, multiple standby nodes in the cluster may be abnormal. In this case, the number of synchronous standby nodes is insufficient for the services on the primary node. As a result, transactions cannot be committed and the customer's production services are blocked.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Based on the current full and incremental rebuilds, **async build** simulates the **WAL Receiver** as the WAL request thread of a normal standby node, replacing the original solution of synchronizing WALs in the rebuild process. In this way, the standby node can still participate in transaction commit during the rebuild.

Currently, the async build mode is controlled by the GUC parameter **async_build** and is enabled by default.

- Application scenarios
  i. Service keepalive and quick recovery after region/AZ-level faults.
  ii. Service jitter prevention during large-scale node restarts.
- Limitations
  i. This feature applies only to full and incremental builds from the primary node to a standby node of the cluster in **quorum** mode.
  ii. This feature takes effect only when there are more than three nodes, more than two synchronous standby nodes, and at least one primary node and one standby node functioning properly.

Performance Metrics

- **Test environment**:
  - Hardware configuration
    - CPU: 64 cores
    - Memory: 256 GB
    - Disk: NVMe SSD RAID group
    - Network: 100G dedicated network (The test network has no upper limit on the transmission bandwidth.)
  - Data model
    - The TPC-C benchmark test data contains 40,000 warehouses, with a total size of about 4.5 TB.
    - Characteristics: The TPC-C model contains multiple wide tables, and a large number of indexes and relationships. Its data files are not stored continuously and compactly on disks. During the rebuild, a large number of random I/Os are generated, which poses high requirements on disk IOPS and latency. **The actual effective speed of transmission is lower than the theoretical sequential read/write bandwidth.**
- **Benchmark**:
  - Time required for full rebuild: The time required is proportional to the data volume. In the preceding environment, the rebuild speed can reach **300 MB/s to 500 MB/s** (affected by the network, disk I/O, encrypted compression, and other configurations).
  - Impact on the performance of the primary node: During the rebuild, the TPC-C TPMC performance of the primary node decreases by about **5% to 15%**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Performance comparison**:

  Table 7-58 Rebuild performance

Rebuild Mode	Parallelism	Compressed	CPU Usage (Standby Node)	Disk I/O Usage (Standby Node)	Rebuild Completion Time (Including the Replay Time)
Full rebuild	1	No	48%	550MB/s	04:20:00
Full rebuild +parallelism	2	No	96%	1100MB/s	02:45:00
Full rebuild +parallelism	5	No	250%	2600MB/s	01:45:00
Full rebuild +parallelism	10	No	500%	4500MB/s	01:25:00
Full rebuild +parallelism +compression	2	Yes	60%	700MB/s	03:10:00
Full rebuild +parallelism +compression	5	Yes	140%	1700MB/s	02:05:00
Full rebuild +parallelism +compression	10	Yes	300%	3500MB/s	01:45:00

  Each time the degree of parallelism increases by one unit, the CPU of the client (standby node) consumes 50% more computing resources, and the computing and storage resource usage of the server (primary node) also

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

increases accordingly. As the degree of parallelism increases, the disk I/O approaches the upper limit, and the performance improvement brought by the rebuild decreases. You need to select the rebuild mode based on the actual service resource consumption.

API Description

Table 7-59 GUC parameters

Parameter	Description	Value Range
parallel_build_thread_num	Specifies the degree of parallelism for parallel rebuild. When the value is **1**, the original single-thread rebuild code is run. When the value is **n**, **n** threads are established to complete data transmission during rebuild.	Value range: an integer ranging from 1 to 10 Default value: **1**
enable_parallel_build_compression	Specifies whether to perform LZ4 compression during parallel data rebuild. This parameter is valid only when the value of **parallel_build_thread_num** is greater than 1.	Value range: **on** and **off**. Default value: **off**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range
async_build	Specifies whether to use the asynchronous build process to rebuild a faulty DN. If this functionality is enabled, the asynchronous process is used for rebuild. If the remaining copies are not the majority, the service write capability can be preferentially restored. If this functionality is disabled, the synchronous logic is used for rebuild. If the remaining copies are not the majority, services can be restored only after data and logs are completely synchronized. After this functionality is enabled, if the standby DN to be rebuilt is required to participate in majority commit, the standby DN to be rebuilt must be in the synchronization list configured by the **synchronous_standby_names** parameter, and at least one DN in the synchronization list must be alive except the DN to be rebuilt.	Value range: **on** and **off**. Default value: **on**

- Parallel build

  For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Sending Server" in *Reference*.
- Asynchronous build

  For details, see "Database GUC Parameters > GUC Parameters > HA Replication > Primary Server" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-60 Build parameters

Parameter	Description	Value Range
-b, --mode=MODE	Specifies the mode of a standby DN to be rebuilt.	Value range of **mode**: • **full**: fully resynchronizes the data directory of the primary DN using an image. • **incremental**: incrementally restores the standby DN based on the differences between the data of primary and standby DNs parsed from Xlogs. • **auto** (default value): prefers incremental rebuild. gs_ctl determines whether to continue incremental or full rebuild based on whether incremental rebuild can be performed after the rebuild fails. In the following scenarios, the gs_ctl tool selects the full rebuild mode to restore DNs: – The differential logs of the primary or standby node have been recycled or damaged. – The control file (that is, *$PGDATA*/**global/pg_control**) that records the checkpoint information on the primary or standby node is lost or damaged. – The system identifiers are inconsistent. – The status file (that is, *$PGDATA*/**global_rewind_info**) of incremental

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range
		rebuild is lost or damaged. • **standby_full**: uses a standby node to rebuild another standby node or uses the main standby node in a DR scenario to rebuild the faulty standby node of the standby database instance for DR. The status of the data source standby node or main standby node must be **Normal**. • **standby_incremental**: uses a standby node to incrementally rebuild another standby node or uses the main standby node in a DR scenario to incrementally rebuild the faulty standby node of the standby database instance for DR. The status of the data source standby node or main standby node must be **Normal**. • **standby_auto**: performs rebuild from the standby node first. The gs_ctl tool determines whether to continue the incremental rebuild or perform full rebuild based on whether incremental rebuild can be performed after the rebuild fails. In the following scenarios, the gs_ctl tool selects the full rebuild mode to restore DNs: – The differential logs of the primary or standby node have been recycled or damaged.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range
		– The control file (that is, *$PGDATA*/**global/pg_control**) that records the checkpoint information on the primary or standby node is lost or damaged. – The system identifiers are inconsistent. – The status file (that is, *$PGDATA*/**global_rewind_info**) of incremental rebuild is lost or damaged. • **cross_cluster_full**: performs cross-database instance full build based on shared storage in the intra-city dual-DC scenario. • **cross_cluster_incremental**: performs cross-database instance incremental build based on shared storage in the intra-city dual-DC scenario.
-D, --pgdata=DATADIR	Specifies the data directory.	The value of **DATADIR** must be a valid data directory.
-Z NODE-TYPE	Specifies the running mode of the database that is connected to.	The value of **NODE-TYPE** can be **coordinator** or **datanode**.
-C CONNECTOR	Specifies the connection string used for connecting to the source DN to rebuild a DN on the current node. Only the connection from a standby node to another standby node is supported.	The connection string contains the IP address and port number of the DN to be rebuilt on the current node and those of the source DN. Its format is as follows: 'localhost=Local DN IP localport=Local DN Port+1 remotehost=DN IP remoteport=DN Port+1'

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range
-M	**-M** must be followed by the **SERVERMODE** parameter, which specifies the start mode of a database upon startup.	The value of **SERVERMODE** can be one of the following three: • **primary**: The localhost starts as the primary node. • **standby**: The localhost starts as the standby node. • **pending**: The localhost is pending to be promoted to primary or demoted to standby. • **hadr_main_standby**: The localhost starts as the main standby node. • **cascade_standby**: The localhost starts as a cascaded standby node.
-q	The process does not automatically start after the build is complete. It needs to be started by the caller.	-
-r, --recvtimeout=INTERVAL	Specifies the timeout interval that a standby node waits for receiving logs synchronized from the primary node.	• Value range: an integer. • Default value: **120**, in seconds.
-t, --timeout=SECS	Maximum seconds for waiting the completion of database startup or shutdown.	• Value range: an integer. • Default value: **60**, in seconds.
-U USERNAME	Specifies the replication username used for rebuilding the main standby node of the standby cluster in streaming DR mode across clusters.	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Value Range
-P PASSWORD	Specifies the replication user password used for rebuilding the main standby node of the standby cluster in streaming DR mode across clusters.	-
-Q	After the rebuild is complete, the shared disk flushing is performed. This operation is involved only in the standby cluster for Dorado DR.	• **copy_from_local**: copies data from the localhost to the shared storage. • **copy_from_share**: copies data from a shared storage device to a local storage device. • **force_copy_from_local**: forcibly copies data from the localhost to the shared storage.

7.32.3 Operation Guide

Prerequisites

- **Software and hardware requirements**:
  - The primary and standby nodes can communicate with each other, and the database port and HA channel port have been enabled on the firewall.
  - The disk space of the standby node is sufficient to store at least one complete data copy.
- **Dependencies**:
  - The GaussDB software versions of the primary and standby nodes must be the same.
  - The primary node is normal.
- **Permission requirements**: The OS user who performs the rebuild operation must have the read and write permissions on the data directories of the primary and standby nodes and the database administrator permissions.

Syntax

```
gs_ctl build -D DATADIR -Z NODE-TYPE [-b MODE] [-M SERVERMODE]
```

Precautions

- **High-risk operation**: During the rebuild, the old data on the target standby node will be **completely cleared**. Ensure that the operation objects are correct to avoid misoperations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Impact on performance**: The rebuild operation consumes a large number of network and storage resources on the primary node. Therefore, **perform this operation during off-peak hours or in the maintenance window**.
- **Monitoring**: The rebuild process takes a long time. You are advised to continuously monitor the cluster status, network traffic, and primary node performance after the operation.

Procedure

The following demonstrates the process of manually triggering a rebuild after the standby node (**dn_6002**) is faulty.

Step 1 Log in to the target node.

Log in to the target node as a database user (non-**root** user).

If the database process is deployed in a sandbox or container, access the corresponding sandbox or container.

Step 2 Query information about the target node.

```
cm_ctl query -Cvid
```

Check whether the target node is abnormal (the status is not **Normal**) based on the node status.

Determine the data directory path of the target node based on the node name. For example, **/var/lib/engine/data1/data/dn_6002** (the following steps use this path as an example).

Step 3 Run the rebuild command.

1. Perform an automatic rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b auto -M standby
   ```
2. Perform a full rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b full -M standby
   ```
3. Perform an incremental rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b incremental -M standby
   ```
4. Use the standby node as the server for automatic rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b standby_auto -M standby
   ```
5. Use the standby node as the server for full rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b standby_full -M standby
   ```
6. Use the standby node as the server for incremental rebuild.
   ```
   gs_ctl build -D /var/lib/engine/data1/data/dn_6002 -Z datanode -b standby_incremental -M standby
   ```

Step 4 Monitor the rebuild progress.

```
gs_ctl querybuild -D /var/lib/engine/data1/data/dn_6002
```

Step 5 Verify the rebuild result.

After the status of the standby node becomes **Normal**, connect to the primary node and run the following command to check whether the standby node is in the synchronous streaming replication state:

1. Quorum
   ```
   SELECT application_name, state, sync_state, sync_priority FROM pg_stat_replication;
   ```
   If the standby node is displayed in the query result and its **state** is **Streaming**, the rebuild is successful and the synchronization is normal.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. DCF

```
SELECT channel, peer_role, peer_dcf_role, peer_state FROM gs_paxos_stat_replication;
```

If the standby node of the corresponding IP address is displayed in the query result and its **peer_state** is **Normal**, the rebuild is successful and the synchronization is normal.

----End

7.32.4 Typical Issues

- Q1: The rebuild task fails to be started, and the error message "could not connect to server: Connection timed out" is displayed.
  - **Symptom**: After the **gs_ctl build** command is executed, a connection timeout error is returned after a long time of waiting.
  - **Cause analysis**:
    i. The primary and standby nodes are disconnected.
    ii. The **listen_addresses** of the primary node is not set to the IP address that can be accessed by the standby node or *****.
    iii. The firewall or security group rules block the communications of the database port or streaming replication port.
  - **Solution**:
    i. Run the **ping** and **telnet <primary_ip> <port>** commands to check the network connectivity.
    ii. Check the **listen_addresses** configuration in the **gaussdb.conf** file on the primary node.
    iii. Check the firewall rules and configure the firewall to allow inbound traffic.
- Q2: During the rebuild, the performance of the primary node deteriorates sharply, affecting services.
  - **Symptom**: After the rebuild starts, the query and transaction response time of the primary database increases.
  - **Cause analysis**: Full data reading and sending occupy a large number of disk I/O and network I/O resources.
  - **Solution**:
    i. **Emergency handling**: Suspend the rebuild task. (You can kill the process to suspend the task. Exercise caution when performing this operation.)
    ii. **Workaround**: Re-execute the rebuild task during off-peak hours.
    iii. **Long-term optimization**: Upgrade the network bandwidth or use storage media with higher performance.
- Q3: The rebuild progress is far slower than expected.
  - **Symptom**: The rebuild takes a long time, and the monitoring result shows that the network or disk usage is not high.
  - **Cause analysis**:
    i. The network bandwidth is insufficient or fluctuates.
    ii. The disk I/O of the primary or standby node has a bottleneck (for example, slow disks or RAID degradation).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

iii. The service load on the primary node is extremely high, and CPU resources are heavily contended, affecting the scheduling of the Data Sender process.

– **Solution**:

i. Use tools such as iostat -x 1 and nethogs to check whether the issue is caused by the network or disk bottleneck.

ii. Check the status of the storage system (such as the RAID controller card status and SSD service life).

iii. Perform the rebuild during off-peak hours.

7.33 Upgrade

After installing the first version of a database, if you need to add functions or adjust service logic, you can update the database binary files, system objects, and configuration files while retaining the original database topology and data. You can determine whether to upgrade the existing system based on the new features.

- In-place upgrade: Services need to be stopped, and all nodes in the cluster are upgraded at a time.
- Gray upgrade: All service operations are supported. Some nodes are upgraded first and the system enters the gray observation period. After verifying the stability of the target version, the remaining nodes will be upgraded.
- Rolling upgrade: All service operations are supported.
  - Independent deployment mode: Shards are upgraded by DN. You can upgrade some shards first and then upgrade the remaining shards after verification, or upgrade all shards at a time.
  - Hybrid deployment mode: Shards are upgraded by AZ. You can upgrade some shards first and then upgrade the remaining shards after verification, or upgrade all shards at a time.

7.33.1 In-Place Upgrade (Non-Multi-Primary)

7.33.1.1 Overview

In-place upgrade is an offline upgrade mode. Services must be stopped, and all nodes of the cluster are upgraded at a time. The upgrade tool can be used in two scenarios depending on whether system objects are modified during the upgrade.

- Minor version (binary) upgrade: applies to scenarios such as patch installation and vulnerability fixing.
- Major version upgrade: applies to scenarios such as new feature, function, and module release.

Description

You can stop clusters, update system objects, and switch binary files to implement offline in-place database upgrade.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Scenarios

After the initial version of the database is installed, subsequent addition of new functions or adjustment of service logic can be achieved by updating the database's binary files, system objects and configuration files, with the original database topology and data retained. This is a stable and reliable upgrade mode that ensures service continuity.

Limitations

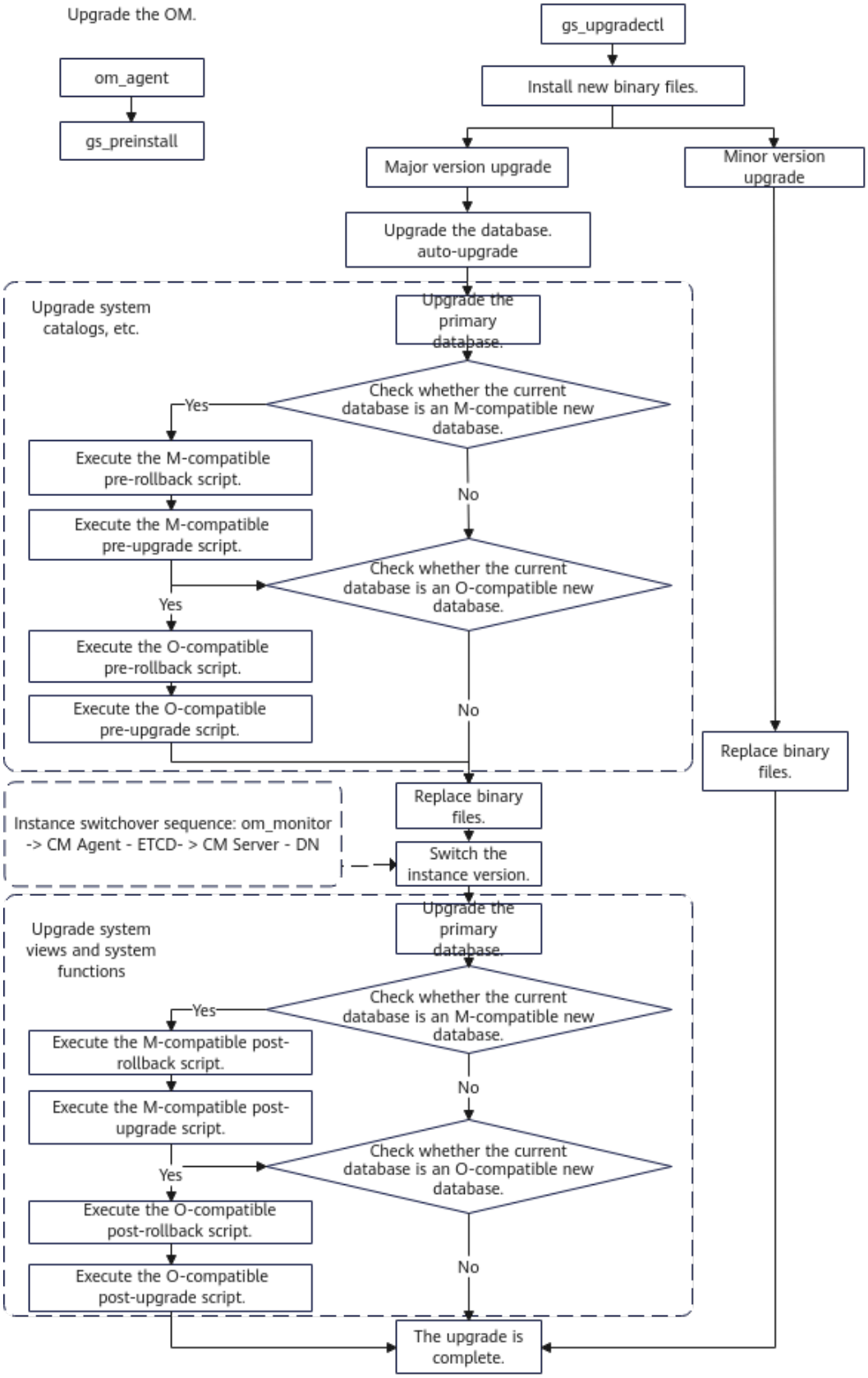
- During the in-place upgrade, the database is started and stopped and is set to read-only. As a result, database services are interrupted.
- All nodes in the cluster are upgraded at a time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Technical Architecture

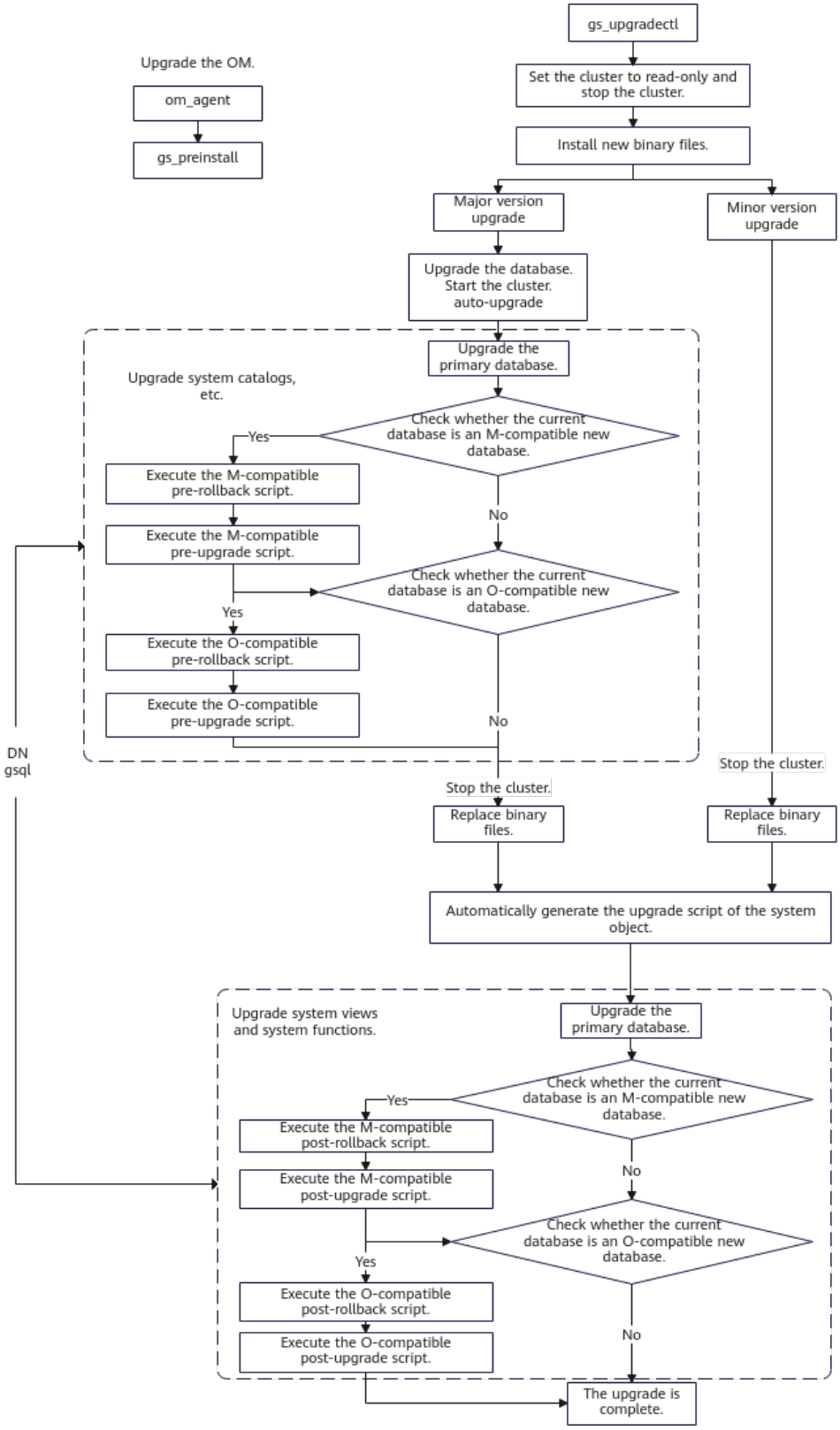
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-66 Technical architecture of in-place upgrade



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-67 Technical architecture of in-place upgrade with the --auto_script parameter specified



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The database upgrade system interconnects with the database O&M platform, using standardized upgrade APIs and a dedicated tool gs_upgradectl, to connect to DNs and gsql. The database is upgraded using upgrade scripts.

If the **auto_script** parameter is specified, the upgrade script is automatically generated after the binary mode is switched. A script is also generated and executed for a minor version upgrade. This parameter is used for internal upgrade transition and is not available on the database O&M platform.

The in-place upgrade rollback is a physical rollback, and the **auto_script** parameter does not take effect.

7.33.1.2 Technical Principle

Working Principle

The following upgrade modes are supported:

1. Minor version upgrade: applies to patch installation or vulnerability fixing. The kernel version number remains unchanged.
2. Major version upgrade: applies to the release of new features or functions. The kernel version number changes.

You can check the kernel version number in the second line of the **version.cfg** file in the upgrade package.

Performance Metrics

1. The deployment mode is three CNs and one DN.
2. The gap between the source and target versions does not exceed two C versions.
3. Services are unloaded.
4. The cluster status, network, and hardware conditions are normal.
5. The total upgrade duration of the cluster is within 20 minutes by default when all nodes are upgraded at a time.

7.33.1.3 Operation Guide

Prerequisites

- Environment:
  - The OSs of all servers are running properly.
  - The network environment is stable.
- Dependency: The **gs_preinstall** script has been successfully executed to preconfigure the environment.
- Permission: The upgrade must be performed by the cluster user specified during the execution of **gs_preinstall**.

Syntax

- Display help information.

```
gs_upgradectl -? | --help
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Display version number.
  ```
  gs_upgradectl -V | --version
  ```
- Select an upgrade policy.
  ```
  gs_upgradectl -t chose-strategy [-l LOGFILE]
  ```
- Upgrade a cluster.
  ```
  gs_upgradectl -t auto-upgrade -X XMLFILE --inplace [-l LOGFILE] [--force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Roll back the upgrade.
  ```
  gs_upgradectl -t auto-rollback -X XMLFILE [-l LOGFILE] [--force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Commit the upgrade.
  ```
  gs_upgradectl -t commit-upgrade -X XMLFILE [-l LOGFILE] [--force_commit] [--skip_rebalance_set]
  ```

  NOTE

  Once the commit is complete, the rollback operation cannot be performed.

Core Limitations

1. Service requirements
   - Services must be stopped during an in-place upgrade.
   - The cluster must meet the following requirements:
     - The data on the primary and standby DNs is completely synchronized.
     - The running status is normal.
2. Operation restrictions
   - It cannot be executed together with node replacement, scale-out, or scale-in.
   - Virtual IP address configuration is not supported.
   - Kerberos is disabled during the upgrade.
   - Roach backup is not supported during in-place upgrade.
   - Do not modify the **version.cfg** file in the installation package.
3. Resource requirements
   - The disk usage of the DN must be less than 85%.
   - The original archive logs become invalid after the upgrade.
   - The original binary directory will be cleared after the upgrade. Do not store personal data in this directory.
4. Parameter control
   - Do not modify the following GUC parameters during the upgrade: **wal_level**, **max_connections**, **max_prepared_transactions**, and **max_locks_per_transaction**.
   - If an exception occurs during the in-place upgrade and the automatic rollback fails as well during the upgrade process, you need to manually execute the rollback command. After the rollback is successful, the GUC parameter settings automatically become invalid.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Upgrade Limitations for Faulty Processes

- Limitations on the upgrade when a minority of VMs or processes are faulty:
  - Prerequisites:
    i. Before delivering the upgrade command, ensure that the cluster is degraded or in a better state.
    ii. The upgrade command containing the **--ignoreVMFaults** option needs to be delivered.
  - Exception handling:
    - During the upgrade, an error will be reported if the VM node of the main process is faulty. This is an expected behavior, and you should deliver the upgrade command to the normal node once again.
    - You can add the **--ignoreVMFaults** option to prevent isolated nodes from being upgraded. After the upgrade is complete, you need to replace the nodes to restore the cluster.
    - The isolated process ignores the error that the process cannot be started.
- Limitations on the upgrade when a majority of processes are faulty:
  - Application scope:
    - Only minor version upgrade is supported. Major version upgrade is not supported.
    - If a fault occurs because the **gaussdb** binary file in the *Installation directory***bin** directory is damaged, a minor version upgrade can be forcibly implemented. If the fault is caused by VM-level faults, the upgrade is not supported.
  - Limitations on forcible upgrade:
    - The upgrade does not provide the restoration function. Therefore, the cluster availability after the upgrade is not assured.
    - Before the forcible upgrade is committed or rolled back, the upgrade cannot be performed even if the cluster is restored.

Performing the In-place Upgrade

Step 1 Log in to a node as user **root**.

Step 2 Go to the **script** directory generated after the installation package is decompressed.

```
cd /opt/software/gaussdb_upgrade/script
```

Step 3 Run the **gs_preinstall** script.

```
./gs_preinstall -U omm -G dbgrp -X /opt/software/GaussDB_Kernel/clusterconfig.xml --alarm-type=1 --non-interactive
```

In the preceding command, **clusterconfig.xml** is the installation configuration file.

Step 4 Switch to user **omm** (cluster user).

```
su - omm
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 5 If the cluster is normal, run the following command to perform the upgrade:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --inplace
```

Step 6 If the database is normal, run the following command to automatically generate the upgrade script and perform the upgrade:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --inplace --auto_script
```

Step 7 If a minority of VMs or processes are faulty, run the following command to perform the upgrade:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --inplace --
ignoreVMFaults=nodeName1,nodeName2,...
```

NOTE

nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster and the list of nodes with faulty processes.

Step 8 If a majority of processes are faulty because the **gaussdb** binary file is damaged, run the following command to forcibly upgrade the minor version:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --inplace --force
```

Step 9 Query the version after upgrade.

```
gs_ssh -c "gsql -V"
```

NOTE

If the query result is not the target version, perform the rollback and then perform the upgrade again.

Step 10 Check the cluster status for the upgrade.

```
cm_ctl query -Cdvi
```

If the value of **Cluster State** is **normal** in the command output, the cluster is normal.

Step 11 Commit the upgrade.

```
gs_upgradectl -t commit-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

NOTE

Once the commit is complete, the rollback operation cannot be performed.

----End

Rolling Back the In-place Upgrade

Step 1 Run the following command to roll back the kernel code. After the rollback is complete, run the pre-processing command in the old package to ensure that the kernel version is the same as the OM code version, and perform steps **Step 1** to **Step 3**. To perform the upgrade again, perform steps **Step 1** to **Step 3** to run the pre-processing command in the new package.

```
gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If the node to be rolled back or a minority of processes on the node are faulty, run the following command to perform the rollback:
  ```
  gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml --ignoreVMFaults=nodeName1,nodeName2,...
  ```
  nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster or the list of nodes with faulty processes.

  During the rollback, you need to specify the list of faulty nodes to be ignored or the list of nodes with faulty processes.

Step 2 Check the version number after the rollback.

```
gsql -V
```

----End

7.33.1.4 Common Issues

- Symptom:

  The commit ID of the new cluster is the same as that of the old cluster.
- Possible cause:
  a. The upgrade package version is the same as the baseline package version. The upgrade is not allowed.
  b. The commit request is initiated without executing the upgrade, which violates the operation process.
- Solution:
  a. Replace the upgrade package with a valid one that is different from the baseline package version.
  b. Perform the following operations in sequence:
     a. Complete the upgrade process.
     b. Perform the commit operation.

7.33.2 Gray Upgrade (Non-Multi-Primary)

7.33.2.1 Overview

Description

Gray upgrade is an online upgrade mode that preferentially upgrades some nodes. You can choose to upgrade partial nodes first. After the upgrade, enter the gray observation period, and check whether the target version is running properly. Then, upgrade the remaining nodes to complete the upgrade of all nodes.

The core implementation includes the following three dimensions:

- Compatibility for system catalogs and functions
  - For system catalog definition changes caused by major version upgrades, the system catalog definitions and system functions of the source and target versions are embedded in binary files.
  - Ensure that the binary files of the source and target versions can parse and use the system catalog tuples of the source and target versions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Binary file replacement policy
  - For the replacement of source and target binary files involved in major version upgrades and binary upgrades, the binary files on the specified node are replaced in gray mode first, and the system running stability is checked.
  - After the system runs stably for a period of time, the binary files on the remaining nodes are replaced.
- OS and hardware upgrade collaboration
  - Scenario where the OS or hardware upgrade is involved:

    Before performing gray upgrade on some nodes, you need to switch the primary instance of the target node to a non-gray node.
  - Scenario where only the database binary upgrade is involved:
    - To minimize the impact on services, the coexistence of two binary files on the same node is used. A soft link is used to switch between the two binary files to implement the process version upgrade.
    - Services are interrupted for no more than 10 seconds when all nodes are upgraded. If the **-g** or **-h** mode is used, services are interrupted for no more than 10 seconds when some nodes are upgraded at a time.

Scenarios

All service operations are supported. Node-level rolling gray upgrade can be performed based on the user-defined gray upgrade sequence. You can upgrade the standby DN first. Then, the primary DN can be switched over, allowing you to control the switchover time.

Limitations

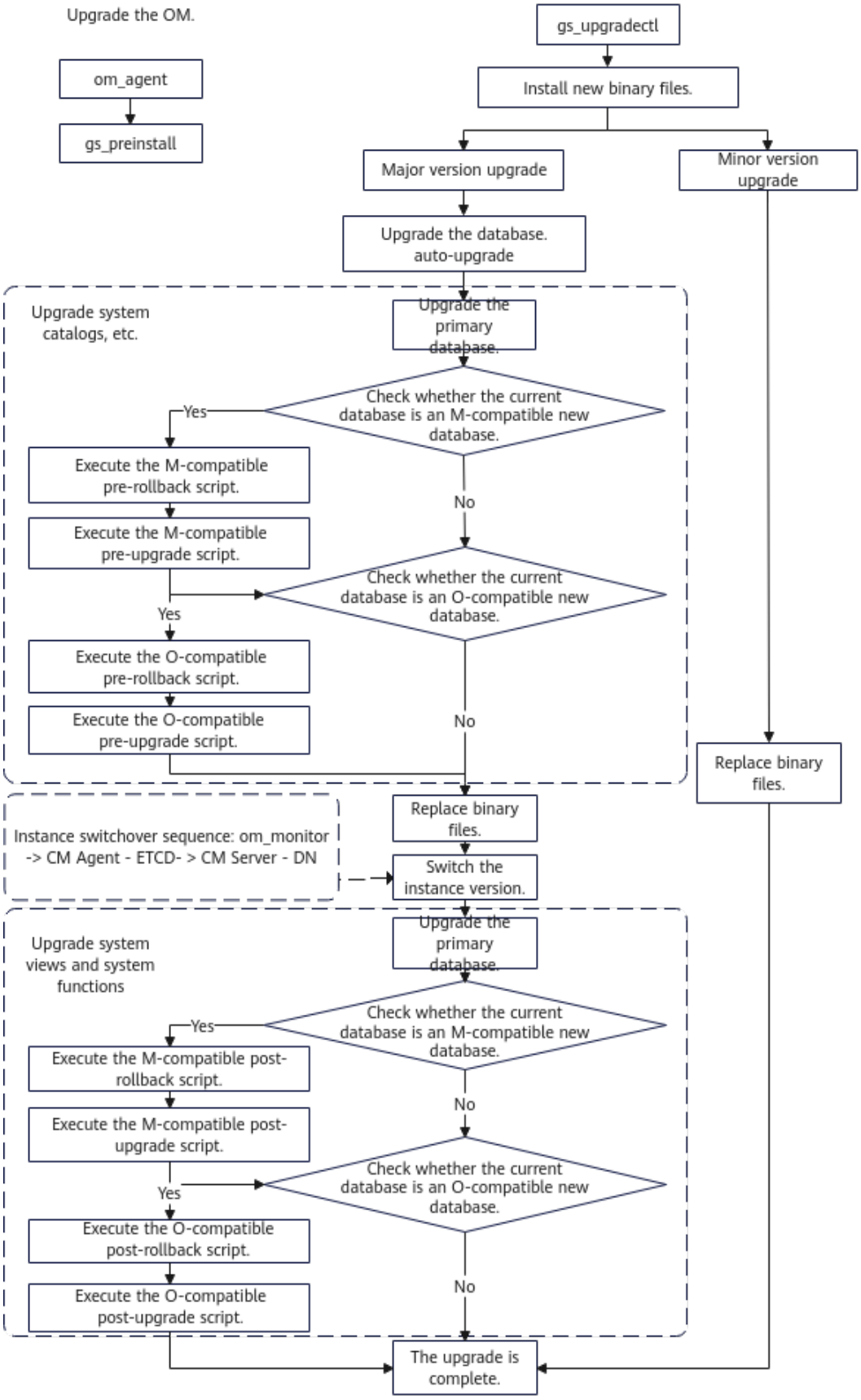
1. Deployment requirements
   - Supported mode: Three CNs and one DN.
   - Version gap: The gap between the source and target versions does not exceed two C versions.
2. Environment conditions
   - Service load: The service must be unloaded during the upgrade.
   - System status: The cluster health status, network, and hardware environment must meet the running standards.
3. Performance metrics

   The total upgrade duration of the database is within 20 minutes by default when all nodes are upgraded at a time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

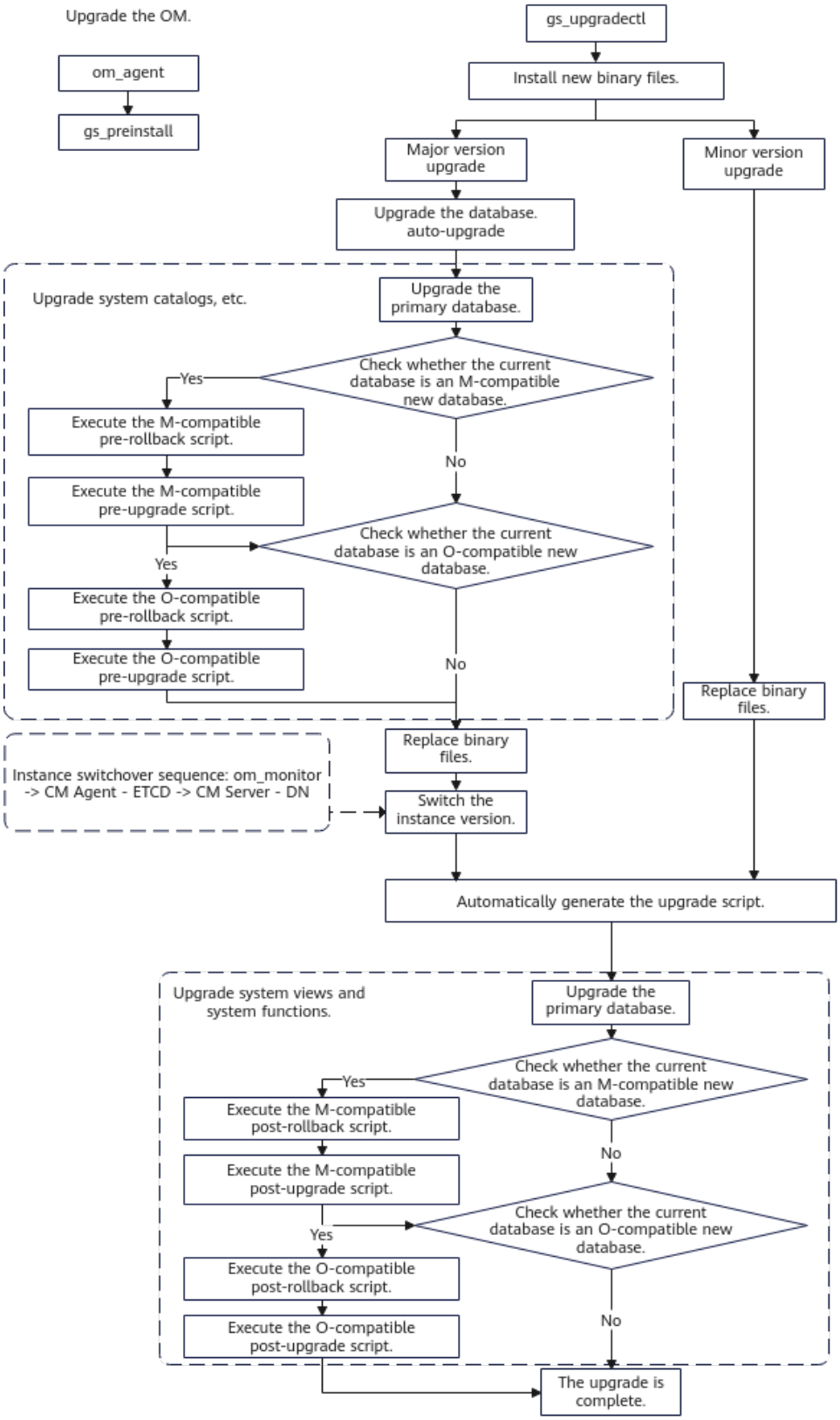
Technical Architecture

Figure 7-68 Technical architecture of gray upgrade



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-69 Technical architecture of gray upgrade with the --auto_script parameter specified



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The database upgrade system interconnects with the database O&M platform, using standardized upgrade APIs and a dedicated tool gs_upgradectl, to connect to DNs and gsql. The database is upgraded using upgrade scripts.

If the **auto_script** parameter is specified, the upgrade script is automatically generated after the binary mode is switched. A script is also generated and executed for a minor version upgrade. This parameter is used for internal upgrade and is not open to the database O&M platform.

To minimize the impact of binary file replacement on services, the following phased binary file replacement policies are used:

1. Switch the process management components om_monitor and cm_agent.
2. Switch the cluster management components cm_server and ETCD that do not affect services.
3. Switch the DN, CN, and GTM components that affect services.

7.33.2.2 Technical Principles

Core Mechanism

The gray upgrade can be implemented in phases to achieve the following objectives:

- Upgrading standby DNs first: Upgrade some standby DNs first. If the upgrade fails, service continuity is not affected.
- Upgrading some CN nodes: Upgrade some CN nodes and route services to other CN nodes that have not been upgraded to ensure that services are not interrupted during the upgrade.
- Upgrading components by service impact: Upgrade the component nodes (such as ETCD and CM Server) that have small impact on services first. Even if the upgrade fails, the impact on services can be minimized.
- Observation period verification mechanism: After each batch of nodes are upgraded, the upgrade observation period is provided to verify the upgrade status and dynamically evaluate upgrade risks.

Performance Metrics

1. Services are interrupted for no more than 10 seconds when all nodes are upgraded.
2. If the **-g** or **-h** mode is used, services are interrupted for no more than 10 seconds when some nodes are upgraded at a time.
3. In the deployment mode involving three CNs and one DN, the gap between the source and target versions does not exceed two C versions, services are unloaded, the database instance status, network, and hardware conditions are normal, the total upgrade duration of the instance is within 20 minutes by default when all nodes are upgraded at a time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.33.2.3 Operation Guide

Prerequisites

- Environment:
  - The OSs of all servers are running properly.
  - The network environment is stable.
- Dependency: The **gs_preinstall** script has been successfully executed to preconfigure the environment.
- Permission: The installation must be performed by the cluster user specified during the execution of **gs_preinstall**.

Syntax

- Display help information.
  ```
  gs_upgradectl -? | --help
  ```
- Display version number.
  ```
  gs_upgradectl -V | --version
  ```
- Select an upgrade policy.
  ```
  gs_upgradectl -t chose-strategy [-l LOGFILE]
  ```
- Upgrade a cluster.
  ```
  gs_upgradectl -t auto-upgrade -X XMLFILE {-g NODENUMBER | -h HOSTNAME | --continue} [-l LOGFILE] [--rolling]
  [--grey] [--begin=ShardStartNumber] [--end=ShardEndNumber] [--force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Roll back the upgrade.
  ```
  gs_upgradectl -t auto-rollback -X XMLFILE [-l LOGFILE] [--force] [--hadr_force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Commit the upgrade.
  ```
  gs_upgradectl -t commit-upgrade -X XMLFILE [-l LOGFILE] [--force_commit] [--skip_rebalance_set]
  ```

  NOTE

  Once the commit is complete, the rollback operation cannot be performed.

Core Limitations

1. Service requirements
   - Gray upgrade supports all service operations.
   - The cluster must meet the following requirements:
     - The data on the primary and standby DNs is completely synchronized.
     - The running status is normal.
   - You are advised to enable flow control when the service load is heavy.
2. Operation restrictions
   - The operation cannot be executed together with node replacement, scale-out, or scale-in.
   - Virtual IP address configuration is not supported.
   - Kerberos is disabled during the upgrade.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Do not modify the **version.cfg** file in the installation package.
- If gray upgrade is selected, node replacement, IP address change, scale-out, scale-in, and hot patch installation are not supported during the upgrade. In addition, no restoration is recommended and no CN can be added if only some nodes are upgraded successfully.

3. Resource requirements
   - The disk usage of the DN must be less than 85%.
   - The original archive logs become invalid after the upgrade.
   - The original binary directory will be cleared after the upgrade. Do not store personal data in this directory.
4. Parameter control
   - Do not modify the following GUC parameters during the upgrade: **wal_level**, **max_connections**, **max_prepared_transactions**, and **max_locks_per_transaction**.
   - If the grade upgrade fails due to an exception in the process, you need to manually roll back the upgrade. The next upgrade can be performed only after the rollback is successful. After the rollback is successful, the GUC parameters set during the upgrade become invalid.
5. Interaction between Roach backup and upgrade:
   - Roach backup is supported during the gray upgrade observation period.
   - When gray upgrade is only executed on some nodes, the major versions of the nodes involved in backup must be the same. Otherwise, the backup fails. For disk backup, all nodes are verified. For OBS, NAS, and XBSA backup, the nodes to be backed up are verified. The rules for selecting the nodes to be backed up are as follows:
     - Nodes to which the backup command is delivered
     - GTM: primary GTM
     - CM Server: primary CM Server
     - DN: primary DN in each shard for backup on the primary node and fastest standby DN in each shard in replay for backup on the standby node
     - CN: first normal CN if archiving is disabled, or all normal CNs if archiving is enabled

Upgrade Limitations for Faulty Processes

- Limitations on the upgrade when a minority of VMs or processes are faulty:
  - Prerequisites:
    i. Before delivering the upgrade command, ensure that the cluster is degraded or in a better state.
    ii. The upgrade command containing the **--ignoreVMFaults** option needs to be delivered.
  - Exception handling:

        - During the upgrade, an error will be reported if the VM node of the main process is faulty. This is an expected behavior, and you should deliver the upgrade command to the normal node once again.
        - You can add the **--ignoreVMFaults** option to prevent isolated nodes from being upgraded. After the upgrade is complete, you need to replace the nodes to restore the cluster.
        - The isolated process ignores the error that the process cannot be started.
    - Limitations on the upgrade when a majority of VMs or processes are faulty:
        - Application scope:
            - Only minor version upgrade is supported. Major version upgrade is not supported.
            - If a majority of processes are faulty because the **gaussdb** binary file in the *Installation directory***bin** directory is damaged, a minor version upgrade can be forcibly implemented. If the faulty processes are caused by VM-level errors, the upgrade is not supported. A minor version upgrade is one during which the kernel version number remains unchanged. Otherwise, the upgrade is a major version upgrade. You can check the kernel version number in the second line of the **version.cfg** file in the upgrade package.)
        - Limitations on forcible upgrade:
            - The upgrade does not provide the restoration function. Therefore, the cluster availability after the upgrade is not assured.
            - Before the forcible upgrade is committed or rolled back, the upgrade cannot be performed even if the cluster is restored.
            - Rollback is supported. After the rollback is successful, the cluster availability is not assured.

Performing the Gray Upgrade

Step 1 Log in to a node as user **root**.

Step 2 Go to the **script** directory generated after the installation package is decompressed.

```
cd /opt/software/gaussdb_upgrade/script
```

Step 3 Run the **gs_preinstall** script.

```
./gs_preinstall -U omm -G dbgrp  -X /opt/software/GaussDB_Kernel/clusterconfig.xml  --alarm-type=1 --non-interactive
```

In the preceding command, **clusterconfig.xml** is the installation configuration file.

Step 4 Switch to user **omm** (cluster user).

```
su - omm
```

Step 5 Gray upgrade can be performed in the following modes. To specify a node, use the **-h** mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1. Upgrade a specified node first, and then upgrade other nodes. For details, see **Step 6**, **Step 7**, and **Step 9**.
2. Upgrade the specified number of nodes first, and then upgrade other nodes. For details, see **Step 8** and **Step 9**.

Step 6 Perform gray upgrade on one specified node **pekpexper00001**.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -h pekpexper00001
```

NOTE

- If the node to be upgraded or a minority of processes on the node are faulty, run the following command to perform the upgrade:
  ```
  gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -h pekpexper00001 --ignoreVMFaults=nodeName1,nodeName2,...
  ```
  =nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster or the list of nodes with faulty processes.

  During the initial gray upgrade, you need to specify the list of faulty nodes to be ignored or the list of nodes with faulty processes. You do not need to specify it again during the subsequent gray upgrade unless there are new faulty nodes and faulty process nodes.

Step 7 Perform gray upgrade on three specified nodes **pekpexper00002**, **pekpexper00003** and **pekpexper00004**.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -h pekpexper00002,pekpexper00003,pekpexper00004
```

After the upgrade is successful, go to **Step 9** to perform upgrade on other nodes.

Step 8 Perform gray upgrade on three nodes. (If you perform gray upgrade on some nodes in a cluster, the number of nodes specified by **-g** cannot exceed 5. If you perform gray upgrade on all nodes in a cluster, the number of all nodes are specified by **-g**.)

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -g 3
```

After the upgrade is successful, go to **Step 9** to perform upgrade on other nodes.

Step 9 On the node that is successfully upgraded in **Step 8**, upgrade all the remaining nodes in the database instance.

- Use the **--continue** parameter to upgrade all remaining nodes in the database instance.
  ```
  gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --continue
  ```
- Use the **--continue** and **--auto_script** parameters to upgrade all remaining nodes in the database instance.
  ```
  gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --continue --auto_script
  ```

Step 10 Upgrade the cluster. In independent deployment mode, gray upgrade by shard can be performed. Run the following upgrade command to specify multiple shards at a time.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=ShardStartNumber --end=ShardEndNumber --grey
```

To upgrade the first shard, run the following command:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=1 --end=1 --grey
```

In hybrid deployment mode, gray upgrade by AZ can be performed. You can run the following command to upgrade all nodes in any AZ:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -h nodename1,nodename2... --rolling --grey
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Note: *nodename* in the command must be all nodes in one or more AZs. If all nodes in the cluster are upgraded at a time and the total number of nodes is *n*, you can run the following combined command:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -g n --rolling --grey
```

Step 11 Use the **--auto_script** parameter to automatically generate an upgrade script and perform gray upgrade on all three nodes.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -g 3 --auto_script
```

Step 12 Query the version after upgrade.

```
gs_ssh -c "gsql -V"
```

NOTE

If the query result is not the target version, perform the rollback and then perform the upgrade again.

Step 13 Check the cluster status for the upgrade.

```
cm_ctl query -Cdvi
```

If the value of **Cluster State** is **normal** in the command output, the cluster is normal.

Step 14 Commit the upgrade.

```
gs_upgradectl -t commit-upgrade  -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

NOTE

Once the commit is complete, the rollback operation cannot be performed.

----End

Rolling Back the Gray Upgrade

Step 1 Run the following command to roll back the kernel code. After the rollback is complete, run the preinstall command in the old package by following steps **Step 1** to **Step 3** to ensure that the kernel code is the same as the OM code To perform the upgrade again, perform steps **Step 1** to **Step 3** to run the preinstall command in the new package.

```
gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

NOTE

- If the node to be rolled back or a minority of processes on the node are faulty, run the following command to perform the rollback:

  ```
  gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml --ignoreVMFaults=nodeName1,nodeName2,...
  ```

  =nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster or the list of nodes with faulty processes.

  During the rollback, you need to specify the list of faulty nodes to be ignored or the list of nodes with faulty processes.

Step 2 Run the following command to check the version number after the rollback:

```
gsql -V
```

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.33.2.4 FAQ

- Q: When the upgrade from version A to version B is not committed, the commit ID verification is triggered upon further upgrading to version C.
- A: Commit the upgrade from version A to version B before upgrading from version B to version C.

7.33.3 Rolling Upgrade

7.33.3.1 Overview

Description

For the rolling upgrade, you can upgrade certain DN or AZ shards first and check whether the upgraded AZs are running properly. Then, upgrade the remaining AZs. You can also upgrade all shards at a time.

Scenarios

Rolling upgrade supports all service operations. The implementation varies depending on the deployment mode.

- Independent deployment mode: The cluster is rollingly upgraded by DN shard.
- Hybrid deployment mode: The cluster is rollingly upgraded by AZ shard. The entered nodes must be all nodes in one or more AZs.

Limitations

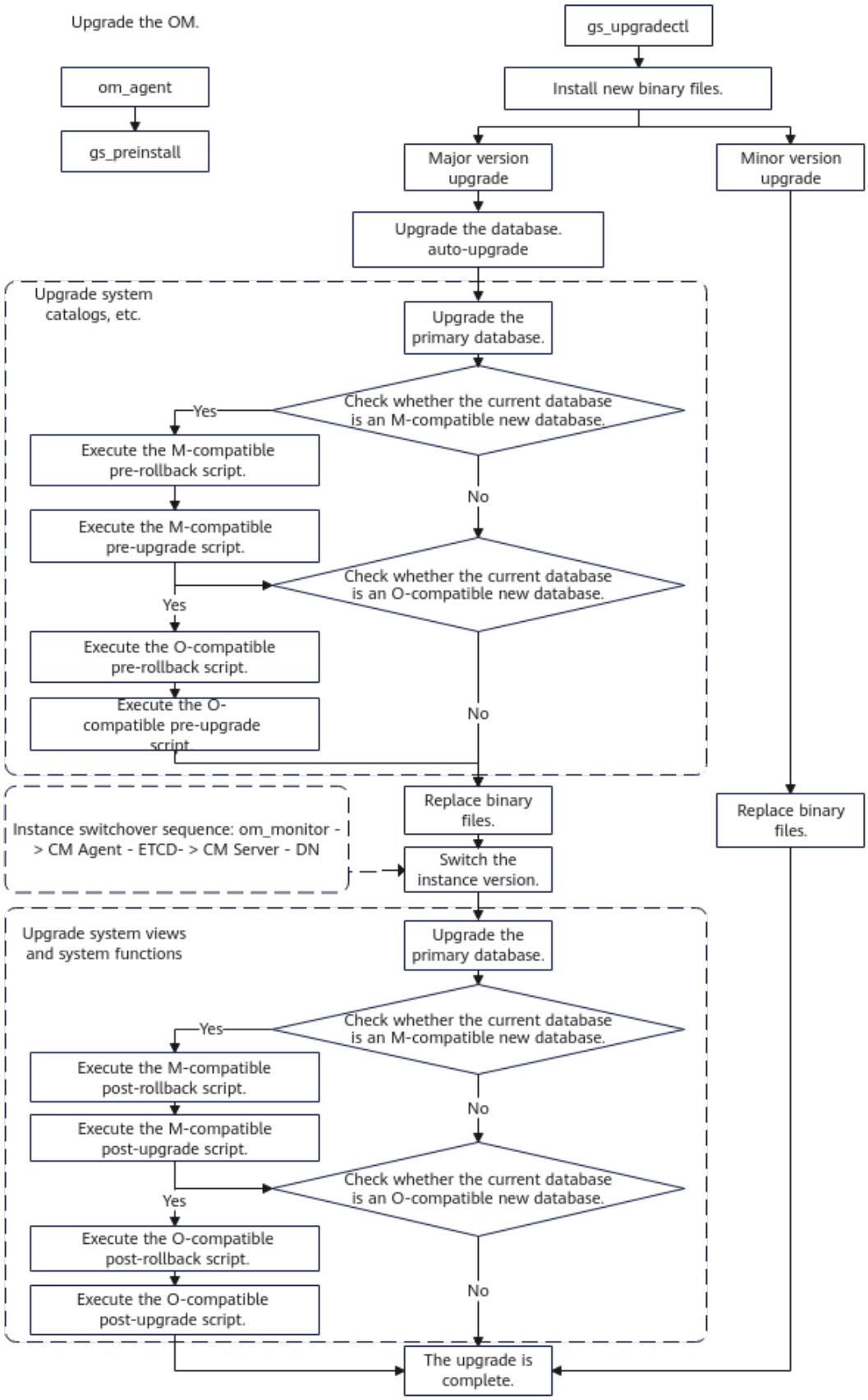
- The rolling upgrade by DN shard requires that the cluster be deployed independently and each instance be deployed on an independent node.
- During the upgrade of each shard, services are interrupted for no more than 10 seconds. Services are interrupted two to three times during the upgrade of the last shard.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.33.3.2 Technical Principle

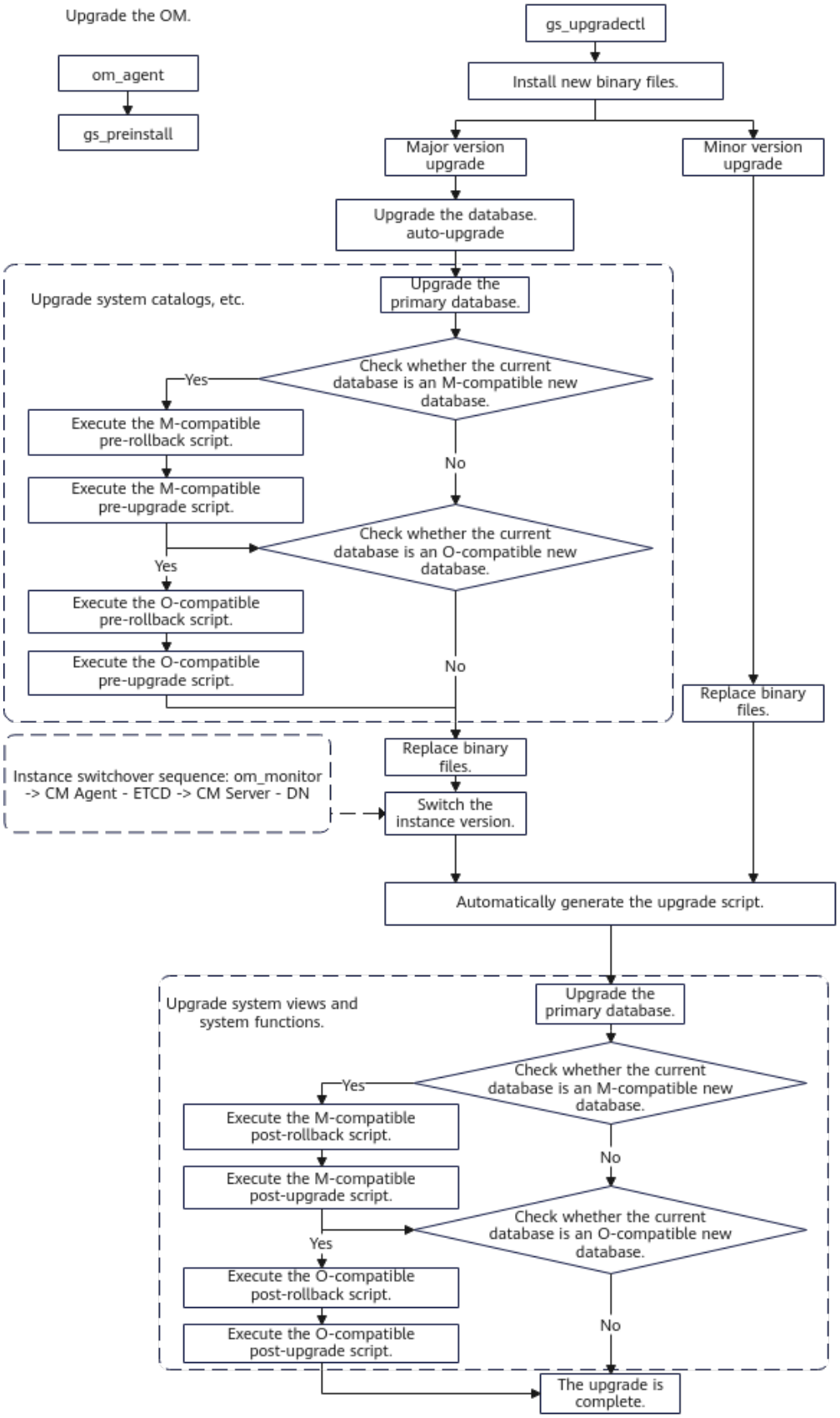
Technical Architecture

Figure 7-70 Technical architecture of rolling upgrade



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-71 Technical architecture of rolling upgrade with the --auto_script parameter specified



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The database upgrade system interconnects with the database O&M platform, using standardized upgrade APIs and a dedicated tool gs_upgradectl, to connect to DNs and gsql. The database is upgraded using upgrade scripts.

If the **auto_script** parameter is specified, the upgrade script is automatically generated after the binary mode is switched. A script is also generated and executed for a minor version upgrade. This parameter is used for internal upgrade and is not open to the database O&M platform.

To minimize the impact of binary file replacement on node services, the following phased binary file replacement policies are used:

1. First, switch the process management components om_monitor and cm_agent.
2. Then, switch the cluster management components cm_server and ETCD that do not affect services.
3. Finally, switch the DN, CN, and GTM components that affect services.

For instance switchover, different rolling modes are used for different instances.

- One primary and multiple standby instances: The standby node is switched first, and then the primary node is switched.
- The rolling upgrade mode is as follows:
  - CN: multi-round upgrade. In each round, a single node is switched.
  - GTM/CM Server: two rounds of upgrade. In the first round, all standby nodes are upgraded in batches. In the second round, the primary node is upgraded.
  - DN/ETCD: Multiple rounds of upgrade. Upgrade a minority of nodes, then the remaining minority of nodes, and finally the primary node. The specific policy is as follows: In the first round, two nodes (one for each) are upgraded. If the total number of nodes is an odd number, (n/2) nodes are upgraded. If the total number of nodes is an even number, (n/2 - 1) nodes are upgraded.

Performance Metrics

1. The service interruption duration during the upgrade of a single shard ≤ 10s.
2. Services are interrupted for two to three times when the last shard is upgraded.
3. The deployment mode is three CNs and one DN. The gap between the source and target versions does not exceed two C versions.
4. All-node upgrade duration: Services are unloaded, the cluster status, network, and hardware conditions are normal, the total upgrade duration of the cluster is within 20 minutes by default when all nodes are upgraded at a time.

7.33.3.3 Operation Guide

Prerequisites

- Environment:
  - The OSs of all servers are running properly.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The network environment is stable.
- Dependency: The **gs_preinstall** script has been successfully executed to preconfigure the environment.
- Permission: The installation must be performed by the cluster user specified during the execution of **gs_preinstall**.

Syntax

- Display help information.
  ```
  gs_upgradectl -? | --help
  ```
- Display version number.
  ```
  gs_upgradectl -V | --version
  ```
- Select an upgrade policy.
  ```
  gs_upgradectl -t chose-strategy [-l LOGFILE]
  ```
- Upgrade a cluster.
  ```
  gs_upgradectl -t auto-upgrade -X XMLFILE {-g NODENUMBER | -h HOSTNAME | --continue} [-l LOGFILE] [--rolling]
  [--grey] [--begin=ShardStartNumber] [--end=ShardEndNumber] [--force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Roll back the upgrade.
  ```
  gs_upgradectl -t auto-rollback -X XMLFILE [-l LOGFILE] [--force] [--hadr_force] [--ignoreVMFaults=HOSTNAME][--auto_script]
  ```
- Commit the upgrade.
  ```
  gs_upgradectl -t commit-upgrade -X XMLFILE [-l LOGFILE] [--force_commit] [--skip_rebalance_set]
  ```

  NOTE

  - Once the commit is complete, the rollback operation cannot be performed.

Core Limitations

1. Service requirements
   - The cluster must meet the following requirements:
     - The data on the primary and standby DNs is completely synchronized.
     - The running status is normal.
   - You are advised to enable flow control when the service load is heavy.
2. Operation restrictions
   - It cannot be executed together with node replacement, scale-out, or scale-in.
   - Virtual IP address configuration is not supported.
   - Kerberos is disabled during the upgrade.
   - Do not modify the **version.cfg** file in the installation package.
   - If gray upgrade is selected, node replacement, IP address change, scale-out, scale-in, and hot patch installation are not supported during the upgrade. In addition, no restoration is recommended and no CN can be added if only some nodes are upgraded successfully.
3. Resource requirements
   - The disk usage of the DN must be less than 85%.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The original archive logs become invalid after the upgrade.
  - The original binary directory will be cleared after the upgrade. Do not store personal data in this directory.

4. Parameter control
   - Do not modify the following GUC parameters during the upgrade: **wal_level**, **max_connections**, **max_prepared_transactions**, and **max_locks_per_transaction**.
   - If the grade upgrade fails due to an exception in the process, you need to manually roll back the upgrade. The next upgrade can be performed only after the rollback is successful. After the rollback is successful, the GUC parameters set during the upgrade become invalid.
5. Interaction between Roach backup and upgrade:
   - Roach backup is supported during the gray upgrade observation period.
   - When gray upgrade is only executed on some nodes, the major versions of the nodes involved in backup must be the same. Otherwise, the backup fails. Disk backup verifies all nodes, while OBS, NAS, and XBSA backup verifies only the nodes involved in the backup. The rules for selecting backup nodes are as follows:
     - Nodes to which the backup command is delivered
     - GTM: primary GTM
     - CM Server: primary CM Server
     - DN: primary DN in each shard for backup on primary and fastest standby DN in each shard in replay for backup on standby
     - CN: first normal CN if archiving is disabled, or all normal CNs if archiving is enabled

Upgrade Limitations for Faulty Processes

- Limitations on the upgrade when a minority of VMs or processes are faulty:
  - Prerequisites:
    i. Before delivering the upgrade command, ensure that the cluster is degraded or in a better state.
    ii. The upgrade command containing the **--ignoreVMFaults** option needs to be delivered.
  - Exception handling:
    - During the upgrade, an error will be reported if the VM node of the main process is faulty. This is an expected behavior, and you should deliver the upgrade command to the normal node once again.
    - You can add the **--ignoreVMFaults** option to prevent isolated nodes from being upgraded. After the upgrade is complete, you need to replace the nodes to restore the cluster.
    - The isolated process ignores the error that the process cannot be started.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Limitations on the upgrade when a majority of VMs or processes are faulty:
  - Application scope:
    - Only minor version upgrade is supported. Major version upgrade is not supported.
    - If a majority of processes are faulty because the **gaussdb** binary file in the *Installation directory***bin** directory is damaged, a minor version upgrade can be forcibly implemented. If the faulty processes are caused by VM-level errors, the upgrade is not supported. (A minor version upgrade is one during which the kernel version number remains unchanged. Otherwise, the upgrade is a major version upgrade. You can check the kernel version number in the second line of the **version.cfg** file in the upgrade package.)
  - Limitations on forcible upgrade:
    - The upgrade does not provide the restoration function. Therefore, the cluster availability after the upgrade is not assured.
    - Before the forcible upgrade is committed or rolled back, the upgrade cannot be performed even if the cluster is restored.
    - Rollback is supported. After the rollback is successful, the cluster availability is not assured.

Performing the Rolling Upgrade

Step 1 Log in to a node as user **root**.

Step 2 Go to the **script** directory generated after the installation package is decompressed.

```
cd /opt/software/gaussdb_upgrade/script
```

Step 3 Run the **gs_preinstall** script.

```
./gs_preinstall -U omm -G dbgrp -X /opt/software/GaussDB_Kernel/clusterconfig.xml --alarm-type=1 --non-interactive
```

In the preceding command, **clusterconfig.xml** is the installation configuration file.

Step 4 Switch to user **omm** (cluster user).

```
su - omm
```

Step 5 Upgrade the cluster. In independent deployment mode, the cluster is rollingly upgraded by shard. Run the following upgrade command to specify multiple shards at a time.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=ShardStartNumber --end=ShardEndNumber
```

To upgrade the first shard, run the following command:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=1 --end=1
```

In hybrid deployment mode, the cluster is rollingly upgraded by AZ. You can run the following command to upgrade all nodes in any AZ:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -h nodename1,nodename2... --rolling
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Note: *nodename* in the command must be all nodes in one or more AZs. If all nodes in the cluster are upgraded at a time and the total number of nodes is *n*, you can run the following combined command:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -g n --rolling
```

NOTE

- If the node to be upgraded or a minority of processes on the node are faulty, run the following command to perform the upgrade:
  ```
  gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=1 --end=1 --ignoreVMFaults=nodeName1,nodeName2,...
  ```
  =nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster or the list of nodes with faulty processes.

  During the initial rolling upgrade, you need to specify the list of faulty nodes to be ignored or the list of nodes with faulty processes. You do not need to specify it again during the subsequent rolling upgrade unless there are new faulty nodes and faulty process nodes.

To upgrade the third, fourth, and fifth shards, run the following command:

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml --begin=3 --end=5
```

After the specified shards are upgraded successfully, run the preceding upgrade command to upgrade the remaining shards.

Step 6 Use the **--auto_script** parameter to automatically generate an upgrade script and perform rolling upgrade on all three nodes.

```
gs_upgradectl -t auto-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml -g n --rolling --auto_script
```

Step 7 Query the version after upgrade.

```
gs_ssh -c "gsql -V"
```

Step 8 Check the cluster status for the upgrade.

```
cm_ctl query -Cdvi
```

If the value of **Cluster State** is **normal** in the command output, the cluster is normal.

Step 9 Commit the upgrade.

```
gs_upgradectl -t commit-upgrade -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

NOTE

Once the commit is complete, the rollback operation cannot be performed.

----End

Rolling Back the Rolling Upgrade

Step 1 Run the following command to roll back the kernel code. After the rollback is complete, run the preinstall command in the old package by following steps **Step 1** to **Step 3** to ensure that the kernel code is the same as the OM code To perform the upgrade again, perform steps **Step 1** to **Step 3** to run the preinstall command in the new package.

```
gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If the node to be rolled back or a minority of processes on the node are faulty, run the following command to perform the rollback:
  ```
  gs_upgradectl -t auto-rollback -X /opt/software/GaussDB_Kernel/clusterconfig.xml --ignoreVMFaults=nodeName1,nodeName2,...
  ```
  =nodeName1,nodeName2,... indicates the list of faulty nodes that need to be ignored in the cluster or the list of nodes with faulty processes.

  During the rollback, you need to specify the list of faulty nodes to be ignored or the list of nodes with faulty processes.

Step 2 Run the following command to check the version number after the rollback:

```
gsql -V
```

----End

7.33.3.4 Common Issues

- Q: When the upgrade from version A to version B is not committed, the commit ID verification is triggered upon further upgrading to version C.
- A: Commit the upgrade from version A to version B before upgrading from version B to version C.

7.34 Repairing and Replacing Nodes

7.34.1 Overview

As enterprise data volumes continue to expand and node counts surge, hardware becomes increasingly susceptible to damage. Consequently, physical node replacement and repair have become routine O&M operations. Conventional offline node replacement solutions struggle to guarantee service continuity, and frequent service interruptions cause great losses to customers. Current mainstream database products typically support only limited operations or even interrupt services during node replacement, failing to meet the demands for regular physical node replacement in massive data scenarios. The online node replacement feature addresses these pain points, significantly improving database reliability and delivering continuous, stable data services to users.

Description

If a database instance becomes unavailable due to hardware failures or component exceptions on a node in the database cluster, you can use this feature to repair the database instance.

Scenarios

1. Node replacement (IP address unchanged): Before replacement, users need to adjust the host name and IP address of the new node to match those of the faulty node. This method is suitable for scenarios demanding the same host name and IP address before and after replacement. It requires minimal prerequisites and offers high implementation efficiency.
2. Node replacement (IP address changed): Replacement is performed directly without changing the host name or IP address of the new node. Equivalent to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the cluster warm backup solution, this method allows replacing a running node with a new node while preserving the new node's IP address and host name. This is suitable for scenarios not demanding the same host name and IP address before and after replacement or when a high-performance node is replaced by a lower-performance one. However, this method involves more prerequisites.

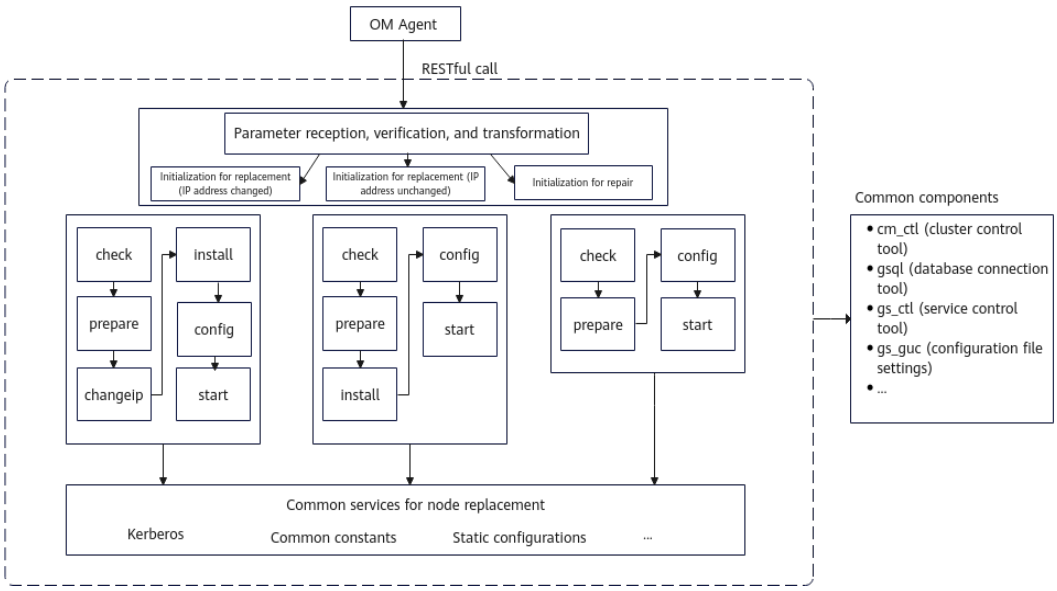
3. Node repair: This method involves reinstalling the faulty components and rebuilding data to quickly repair the faulty node.

Specifications and Limitations

- This feature is not supported in distributed single-node systems.
- Node repair and replacement must be initiated on a healthy node. The system must ensure that only one operation process exists at a time; parallel operations on multiple nodes are not allowed.

Technical Architecture

From the perspective of end-to-end architecture, the node repair and replacement feature provides RESTful APIs to interconnect with the management plane. Common components offer various functionalities for repairing cluster components or modifying the topology structure, as illustrated in the following figure.



7.34.2 Technical Principles

Core Mechanism

Node repair and replacement can be completed online without interrupting services. This is because the database configurations are dynamically loaded and take effect, and the instance can be started properly after data build. The procedure is as follows:

Step 1 (Optional, used for node replacement) The cluster management component CM removes the IP address of the faulty node from the cluster and adds the IP address

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

of the new node to the cluster. All configurations, including updating configuration files and system catalogs, are dynamically loaded and take effect.

Step 2 (Optional, used for node replacement) Create an installation directory and a log directory on the new node, decompress the component binary files, and generate certificates. After this step, all components on the new node are to be repaired.

Step 3 Repair the faulty components, including initializing the components to be repaired, generating configuration files, and performing full/incremental data build.

Step 4 Start the repaired components and verify that all components are in normal state.

----End

7.34.3 Operation Guide

Prerequisites

- Node repair
  - This tool supports node repair for faulty ETCD, DN, CN, CM, and GTM components.
  - When **enable_synclist_single_inst** is set to **off** in a "1 primary DN + multiple standby DNs" setup, a majority of instances in each DN group must remain operational—only a minority of instances are allowed to fail. In such cases, using gs_replace to repair faulty DNs does not affect services. During DN instance repair, the DN ring must contain at least one primary DN that is being properly monitored by CM—this is essential for maintaining data consistency.
  - The cluster must contain at least one operational instance, including the CM Server, GTM, and CN.
  - If ETCD is deployed in the cluster, the number of normal ETCD nodes must exceed half of the total ETCD nodes.
  - During CN instance repair, the cluster must contain at least one normal CN. Otherwise, the repair will fail.
  - If a faulty node is ignored during an upgrade, resulting in a version mismatch between it and normal nodes after the upgrade, direct repair is not permitted. In such cases, use node replacement instead.
  - When multiple CNs are to be repaired, the maximum number of concurrent repairs is constrained by the **max_wal_senders** parameter of normal CNs. Specifically, the number of CNs that can be repaired at a time is half of the **max_wal_senders** value.
- Node replacement (IP address unchanged)
  - When **enable_synclist_single_inst** is set to **off** in a "1 primary DN + multiple standby DNs" setup, a majority of instances in each DN group must remain operational—only a minority of instances are allowed to fail. In such cases, using gs_replace to replace nodes does not affect services. During DN instance repair, the DN ring must contain at least one primary DN that is being properly monitored by CM—this is essential for maintaining data consistency.
  - During DN instance replacement via gs_replace, the replaced DN must not be the primary DN. Otherwise, the replacement will fail.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The cluster must contain at least one operational instance, including the CM Server, GTM, and CN.
  - If ETCD is deployed in the cluster, the number of normal ETCD nodes must exceed half of the total ETCD nodes.
  - The IP address and host name of the new node must be the same as those of the faulty node.
  - A standby DN cannot be replaced by a log node, nor can a log node be replaced by a standby DN.
- Node replacement (IP address changed)
  - When **enable_synclist_single_inst** is set to **off** in a "1 primary DN + multiple standby DNs" setup, a majority of instances in each DN group must remain operational—only a minority of instances are allowed to fail. In such cases, using gs_replace to replace faulty DNs does not affect services. During DN instance repair, the DN ring must contain at least one primary DN that is being properly monitored by CM—this is essential for maintaining data consistency.
  - During DN instance replacement via gs_replace, the replaced DN must not be the primary DN. Otherwise, the replacement will fail.
  - The cluster must contain at least one operational instance, including the CM Server, GTM, and CN.
  - If ETCD is deployed in the cluster, the number of normal ETCD nodes must exceed half of the total ETCD nodes.
  - Users can perform single-node uninstallation to remove a faulty node. This involves uninstalling the node to be replaced or manually cleaning up database-related directories and residual processes on the damaged node.
  - A cluster configuration file must be prepared in advance based on the IP address of the new node. In this file, only the IP address and host name should be adjusted; all other content must remain unchanged.
  - In hybrid deployment scenarios, a standby DN cannot be replaced by a log node, nor can a log node be replaced by a standby DN.

Precautions

- Node repair
  - If the cluster contains faulty CNs in a non-**Deleted** state, DDL statements will trigger an error during repair, although DML statements proceed normally. Other service scenarios remain unaffected.

    NOTE

    The cluster is briefly locked during CN repair. During this period, DDL statements containing explicit transaction start will enter a waiting state. An error will be reported once the cluster is unlocked or the wait time exceeds 20 minutes. For example, temporary table creation will fail with the error "Don't support temp table when need reconnect pooler" after the cluster is unlocked.
  - If binary files are corrupted or missing from the installation directory (*$GAUSSHOME*/**bin/**) of the node hosting a faulty instance, the instance cannot be repaired through instance replacement. In such cases, either copy binary files from a normal node, or uninstall the faulty node and perform node replacement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Node repair operations address all faulty instances on the faulty node.
  - During the config phase, the CM Agent component of the specified node is repaired first to ensure successful retrieval of the node's instance status. Manually stopped instances (if any) on that node will be restarted (not repaired) after successful CM Agent repair. However, if the restart attempt fails within the specified time frame, the instances will undergo repair.
  - Automatic removal of faulty CNs is disabled during instance repair and re-enabled upon repair completion. Therefore, it is advisable to ensure that faulty CNs have been automatically removed (that is, they are in the **Deleted** state) before initiating repair. Otherwise, executing DDL statements during the repair process will trigger an error.
  - If the repair fails due to procedural errors, run the original command again to retry the repair process. Some instances may have been repaired and started but their status has not fully recovered. Continue the repair process until all instances return to normal status.
- Node replacement (IP address unchanged)
  - If the cluster contains faulty CNs in a non-**Deleted** state, DDL statements will trigger an error during repair, although DML statements proceed normally. Other service scenarios remain unaffected. Therefore, it is advisable to ensure that faulty CNs have been automatically removed (that is, they are in the **Deleted** state) before initiating replacement.

    NOTE

    The cluster is briefly locked during CN repair. During this period, DDL statements containing explicit transaction start will enter a waiting state. An error will be reported once the cluster is unlocked or the wait time exceeds 20 minutes. For example, temporary table creation will fail with the error "Don't support temp table when need reconnect pooler" after the cluster is unlocked.
  - If the faulty node and the new node are the same, execute the single-node uninstallation process prior to replacement.
- Node replacement (IP address changed)
  - If the cluster contains faulty CNs in a non-**Deleted** state, DDL statements will trigger an error during repair, although DML statements proceed normally. Other service scenarios remain unaffected. Therefore, it is advisable to ensure that faulty CNs have been automatically removed (that is, they are in the **Deleted** state) before initiating replacement.

    NOTE

    The cluster is briefly locked during CN repair. During this period, DDL statements containing explicit transaction start will enter a waiting state. An error will be reported once the cluster is unlocked or the wait time exceeds 20 minutes. For example, temporary table creation will fail with the error "Don't support temp table when need reconnect pooler" after the cluster is unlocked.
  - If a node experiences complete failure while other nodes have partial instance failures, node replacement (IP address changed) should be performed first. After successful replacement, node repair must still be conducted on the partially failed nodes.
  - If ETCD is deployed in the cluster and the ETCD nodes are replaced sequentially without restarting the primary and standby GTM

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

components, the primary and standby GTM components must be restarted after the last node is replaced to ensure cluster reliability.

Example: Assume ETCD is deployed on node 1, node 2, and node 3 in the cluster. The replacement procedure would be:

- Node 1 fails and is replaced by node 4 without restarting the primary and standby GTM components.
- Node 2 fails and is replaced by node 5 without restarting the primary and standby GTM components.
- Node 3 fails and is replaced by node 6. After this replacement, the primary and standby GTM components must be restarted.

– If a faulty node is disconnected, O&M personnel must manually clean up the following residual items after replacement:

- Database data on the node.
- Residual database processes, instance directories, and environment variables. (The paths to be cleaned must match those configured on the new node. You can refer to the corresponding items on the new node as a reference for cleanup.)

– In scenarios where a standby DN is replaced by a log node, O&M operations such as scale-out, copy addition/reduction, and scale-in can only be performed after all shards have been replaced.

Examples

Example 1: Node repair

This example demonstrates how to repair faulty instances on the **plat1** and **plat2** nodes.

Step 1 Log in to any normal database node as the OS user **omm**.

Step 2 Configure the nodes where instances need repair. This involves cleaning up the space of the instances to be repaired, initializing them, and configuring them.

```
gs_replace -t config -h plat1,plat2
```

If the error "GAUSS_50201: The XXX does not exist." appears, check whether the instance data directory exists. If missing, create it and re-run the command.

NOTE

- If data in the tablespace of a specified node is corrupted due to disk failure, specify the **--force** parameter to forcibly restore the tablespace data after replacing the faulty disk. If **--force** is specified during the config phase, it must also be specified in the start phase.
- During CN instance replacement, do not connect to the nodes for service operations before the CN status returns to **Normal**.
- Before repairing an instance, note that manually configured GUC parameters on the faulty instance and whitelist settings in **gs_hba.conf** will be lost and must be reconfigured afterward.
- The CM isolation feature is automatically disabled during instance repair and will be restored to its pre-replacement state afterward.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 3 Start the nodes where instances need repair.

```
gs_replace -t start -h plat1,plat2
```

Step 4 Reset the instance status. This operation restores the cluster to its initial state with balanced loads across nodes.

> **NOTE**
>
> Before running the **switch** command, ensure that:
>
> - The cluster is in a normal state.
> - All services are stopped.
> - No primary-standby catchup is present. This can be verified using the PG_GET_SENDERS_CATCHUP_TIME view.

```
gs_om -t switch --reset
```

Step 5 Run the **gs_om -t status** command to confirm that the cluster's **balanced** status has returned to **Yes**.

```
[   Cluster State   ]
cluster_state   : Normal
redistributing  : No
balanced        : Yes
```

----End

Example 2: Node replacement (IP address unchanged)

This example assumes that **plat3** is the new node used for replacement.

Step 1 Log in to the new node **plat3** as user **root** and check its character set.

```
echo $LANG
en_US.UTF-8
```

Ensure that the character set and encoding of the new node match those of the other nodes.

If they differ, run the following command to adjust the character set:

```
export LANG="en_US.UTF-8"
```

Step 2 Log in to any normal node in the cluster as user **root**.

Step 3 Access the **script** directory in the decompressed installation package. Assuming the package is located in **/opt/software/gaussdb**, run the following command:

```
cd /opt/software/gaussdb/script
```

Step 4 Prepare the cluster environment.

```
./gs_preinstall -U omm -G dbgrp -X ../clusterconfig.xml --alarm-type=5
```

omm is the OS user running the cluster, **dbgrp** is the group to which the OS user belongs, and **clusterconfig.xml** is the cluster configuration file (retaining the IP address and host name of the original node). This example assumes the file is stored in the same directory as the installation package.

Step 5 Switch to the OS user **omm**.

```
su - omm
```

Step 6 Install the new node.

```
gs_replace -t install -h plat3
```

Step 7 Configure the new node.

```
gs_replace -t config -h plat3
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- During CN instance replacement on a node, do not connect to the node for service operations before the CN status returns to **Normal**.
- Before replacing a node, note that manually configured GUC parameters on the faulty node and whitelist settings in **gs_hba.conf** will be lost and must be reconfigured afterward.
- The CM isolation feature is automatically disabled during node replacement and will be restored to its pre-replacement state afterward.

Step 8 Start the new node.

```
gs_replace -t start -h plat3
```

Step 9 Restore the cluster to its initial load-balanced state.

```
gs_om -t switch --reset
```

Step 10 Run the **gs_om -t status** command to confirm that the cluster's **balanced** status has returned to **Yes**.

```
[   Cluster State   ]
cluster_state   : Normal
redistributing  : No
balanced        : Yes
```

----End

Example 3: Node replacement (IP address changed)

Step 1 Log in to the new node as user **root** and check its character set.

```
echo $LANG
en_US.UTF-8
```

Ensure that the character set and encoding of the new node match those of the other nodes.

If they differ, run the following command to adjust the character set:

```
export LANG="en_US.UTF-8"
```

Step 2 Log in to any normal node in the cluster as user **root**.

Step 3 Access the **script** directory in the decompressed installation package. Assuming the package is located in **/opt/software/gaussdb**, run the following command:

```
cd /opt/software/gaussdb/script
```

Step 4 Prepare the cluster environment.

```
./gs_preinstall -U omm -G dbgrp -X ../clusterconfig.xml --alarm-type=5
```

omm is the OS user running the cluster, **dbgrp** is the group to which the OS user belongs, and **clusterconfig.xml** is the cluster configuration file (updated to the IP address and host name of the new node). This example assumes the file is stored in the same directory as the installation package.

Step 5 Switch to the OS user **omm**.

```
su - omm
```

Step 6 Replace the node.

```
gs_replace -t warm-standby -X clusterconfig.xml
```

If any exception occurs during execution, resolve it based on the prompt information and re-run the command.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- During CN instance replacement on a node, do not connect to the node for service operations before the CN status returns to **Normal**.
- Before replacing a node, note that manually configured GUC parameters on the faulty node and whitelist settings in **gs_hba.conf** will be lost and must be reconfigured afterward.
- The CM isolation feature is automatically disabled during node replacement and will be restored to its pre-replacement state afterward.

Step 7 Restore the cluster to its initial load-balanced state.

```
gs_om -t switch --reset
```

Step 8 Run the **gs_om -t status** command to confirm that the cluster's **balanced** status has returned to **Yes**.

```
[   Cluster State   ]
cluster_state   : Normal
redistributing  : No
balanced        : Yes
```

----End

7.34.4 FAQ

- Q: In hybrid deployment, if the old node to be replaced has the primary DN of a shard, the replacement fails. What should I do?
- A: The node to be replaced cannot have a primary DN. You need to run the **cm_ctl switchover -n nodeId -D datadir** command to switch the primary DN to another node in the same shard, and then continue the replacement.
- Q: What should I do if an instance is faulty, node replacement or node repair?
- A: You are advised to repair the node, because node repair only takes effect on the faulty instance. Normal instances are not affected. Node replacement is mainly used to replace a faulty node with a new node after the server breaks down or the system is faulty. The process is complex.

7.35 Plan Cache

Plan cache is a general mechanism for reusing SQL execution plans. When the same SQL statement is executed, the historical cached execution plan is reused, and the plan does not need to be regenerated. This reduces the overhead of repeatedly generating execution plans and optimizes the performance of executing the same SQL statement. Essentially, it is a mechanism that uses memory to improve performance, which is applicable to the scenario where SQL statements are reused.

You can use specific syntax to specify the use of plan cache. In addition, you can use the GUC parameter **cursor_sharing** to cached plans for SQL statements that are not specified for reuse and reuse the plans. To reuse plans between sessions and reduce the overall memory of the plan cache, the global plan cache (GPC) capability is supported. In addition, the upper limit of the plan cache memory can be controlled to avoid excessive memory usage.

The plan cache functionality can be classified into **prepared plan cache**, **automatic parameterization and plan reuse for identical statements**, **global plan cache**, and **upper limit of the plan cache memory control**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.35.1 Prepared Plan Cache

7.35.1.1 Overview

Description

You can specify the same SQL template statement. When the statement is executed repeatedly, the parsing and plan generation operations can be skipped, and the cached plan generated during the last execution can be reused, thereby improving the execution speed. You can set bound parameters for the SQL statement. Different parameters can be bound to the statement each time they are executed, and the plan template can be reused. The procedure is as follows:

- Preparation phase: Send an SQL statement (with parameters that can be bound) and provide a statement name as the key.
- Execution phase: Specify the statement name key and provide parameter values to execute the prepared and cached SQL statement.

  The bound parameters are optional. You need to specify them only when different parameters are set for the same SQL template. When the SQL query repetition rate is high, the prepared plan cache mode is recommended.

You can use the prepared plan cache functionality through the PreparedStatement method in JDBC or the PREPARE EXECUTE syntax in gsql. For details about how to use JDBC, see **Executing SQL Statements**. For details about how to use PREPARE EXECUTE, see "SQL Reference > SQL Syntax > P > PREPARE" in *Reference*.

In the stored procedure definition, a fixed SQL statement is parsed and a cached plan is generated when the statement is executed for the first time. The plan can be reused.

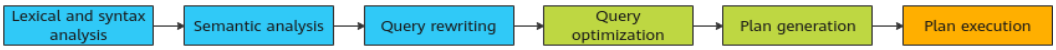
Specifications and Limitations

1. A prepared statement and the corresponding key are at the session level (not globally shared). The key can be used only in the current session to map to the expected plan. If the plan cannot be shared, the plan is private to the session.
2. When a generic plan as a cached plan for reuse, if the GUC parameters that affect only the plan are modified, the modification does not take effect immediately to avoid affecting the plan being reused. When a custom plan is generated, the plan is created each time. The modification of GUC parameters that affect only the plan takes effect immediately.
3. If the object on which the plan cache depends changes (for example, a table is modified by ALTER or refreshed by ANALYZE), the plan cache is marked as invalid due to the change of the metadata of the depended object. After the plan is invalidated, the plan is rebuilt and the system overhead is generated, which may cause performance fluctuation.
4. If the table definition on which the prepared statement depends changes (for example, a column definition is added, deleted, or modified), the expected result definition in the cached plan may not match the actual table definition. As a result, an unexpected result is returned when the statement is executed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.35.1.2 Technical Principle

Core Mechanism



In the SQL execution process, the prepared plan cache can reuse the entire process from lexical and syntax analysis to plan generation. The plan cache reuse effect is affected by the plan type. After the reuse is successful, only the plan execution module needs to be processed each time.

In the preparation phase, the lexical and syntax parsing, semantic analysis, and query rewriting results are cached, and the query optimization and plan generation operations are started. A generic plan or custom plan can be generated. The two types of plans have different features:

- Generic plan: allows the same execution plan to be used for different parameter values.
- Custom plan: fixes parameter values in the plan and requires that a plan be rebuilt for each parameter combination.

If no parameter is bound, the execution of a generic plan is the same as that of a custom plan. By default, the generic plan is selected.

The two types of plans have their own advantages. A generic plan can reduce the optimizer's computing overhead. A custom plan estimates the cost based on specific parameter values to ensure that the execution plan is optimal. For details about the selection mechanism, refer to description of **plan_cache_mode** in "Hint-based Optimization > Hints for Selecting the Custom Plan or Generic Plan" in *SQL Optimization Guide* and "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.

The reuse scope of the plan cache depends on the plan type.

- Generic plan: The intermediate results from lexical and syntax analysis to plan generation are fully reused.
- Custom plan: Only the semantic analysis layer (lexical and syntax analysis, semantic analysis, and query rewriting) is reused.

When the database object on which the plan cache depends changes (for example, the system catalog metadata is modified using DDL statements), the cached plan will not match the new metadata and becomes invalid. In this case, the system automatically triggers the cache invalidation mechanism upon metadata change events, and forcibly rebuilds the plan during the next execution.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

API Description

Table 7-61 Views related to prepared plan cache

View	Description
PG_PREPARED_STATEMENTS	Displays information about all prepared statements that are available in the current session. For details, see "System Catalogs and System Views > System Views > Other System Views > PG_PREPARED_STATEMENTS" in *Reference*.
GS_ALL_PREPARED_STATEMENTS	Displays information about prepared statements that are available in all sessions. For details, see "System Catalogs and System Views > System Views > Other System Views > GS_ALL_PREPARED_STATEMENTS" in *Reference*.

7.35.1.3 Operation Guide

Examples

- JDBC example
  a. Create the prepared statement object **pstmt** by calling the prepareStatement method of the Connection API.
  ```
  PreparedStatement pstmt = conn.prepareStatement("UPDATE customer_t1 SET
  c_customer_name = ? WHERE c_customer_sk = 1");
  ```
  b. Set parameters by calling the setString method of the PreparedStatement API.
  ```
  pstmt.setString(1, "new Data");
  ```
  c. Execute the prepared statement by calling the executeUpdate method of the PreparedStatement API.
  ```
  int rowcount = pstmt.executeUpdate();
  ```
  d. Close the prepared statement object **pstmt** by calling the close method of the PreparedStatement API.
  ```
  pstmt.close();
  ```

  Complete example:

  a. Create a table.
  ```
  gaussdb=# CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
  VARCHAR(32));
  CREATE TABLE
  ```
  b. Run the following JDBC script:
  ```
  import java.sql.Connection;
  import java.sql.PreparedStatement;
  import java.sql.SQLException;
  import java.util.Arrays;
  import java.sql.DriverManager;
  public class TestBatch {
      public static Connection getConnection() throws ClassNotFoundException, SQLException{
   String driver = "com.huawei.gaussdb.jdbc.Driver";
          // Specify the source URL of the database. (Adjust $ip, $port, and database based on the
  actual services.)
          String sourceURL = "jdbc:gaussdb://$ip:$port/database";
          // Obtain the username and password from the environment variables.
          String userName = System.getenv("EXAMPLE_USERNAME_ENV");
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Class.forName(driver);
        return DriverManager.getConnection(sourceURL, userName, password);
    }
    public static void main(String[] args) {
        String sql = "UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1";
        try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql)) {
            conn.setAutoCommit(false);
            pstmt.setString(1, "new Data");
            int rowcount = pstmt.executeUpdate();
            conn.commit();
            pstmt.close();
            conn.close();
        } catch (ClassNotFoundException | SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- JDBC example

  a. Prepare SQL statements for batch execution.

     Use preparedStatement to prepare statements. You can replace the statements with the statements that need to be executed based on the actual service scenario. The following is an example of inserting data into a test table:

     ```
     String sql = "INSERT INTO  TEST_BATCH(v1,v2) VALUES(?,?)";
     PreparedStatement preparedStatement = conn.prepareStatement(sql);
     ```

  b. Bind parameters in batches.

     Use preparedStatement to bind parameters. Then call the addBatch API to add the SQL statements to the batch execution list.

     ```
     for(int i=0;i<5;i++){
        preparedStatement.setString(1,"value1_"+i);
        preparedStatement.setString(2,"value2_"+i);
        preparedStatement.addBatch();
     }
     ```

  c. Execute the batch operation.

     By calling the executeBatch API, preparedStatement executes the batch operation. Upon completion, it returns an array of int results, from which you can determine the number of data records affected by the batch operation.

     ```
     Int[] results = preparedStatement.executeBatch();
     System.out.println(Arrays.toString(results));
     ```

  d. Release resources and close the database connection.

     ```
     preparedStatement.close();
     conn.close();
     ```

  Complete example:

  a. Create a table.

     ```
     gaussdb=# CREATE TABLE TEST_BATCH(
     V1 TEXT,
     V2 TEXT);
     CREATE TABLE
     ```

  b. Run the following Java script:

     ```
     import java.sql.Connection;
     import java.sql.PreparedStatement;
     import java.sql.SQLException;
     import java.util.Arrays;
     import java.sql.DriverManager;
     public class TestBatch {
     ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    public static Connection getConnection() throws ClassNotFoundException, SQLException{
 String driver = "com.huawei.gaussdb.jdbc.Driver";
        // Specify the source URL of the database. (Adjust $ip, $port, and database based on the
actual services.)
        String sourceURL = "jdbc:gaussdb://$ip:$port/database";
        // Obtain the username and password from the environment variables.
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Class.forName(driver);
        return DriverManager.getConnection(sourceURL, userName, password);
    }
    public static void main(String[] args) {
        String sql = "insert into TEST_BATCH(v1,v2) values(?,?)";
        try (Connection conn = getConnection(); PreparedStatement preparedStatement =
conn.prepareStatement(sql)) {
            conn.setAutoCommit(false);
            for (int i = 0; i < 5; i++) {
        preparedStatement.setString(1,"value1_"+i);
        preparedStatement.setString(2,"value2_"+i);
        preparedStatement.addBatch();
            }
            int[] results = preparedStatement.executeBatch();
            conn.commit();
            System.out.println(Arrays.toString(results));
            preparedStatement.close();
            conn.close();
        } catch (ClassNotFoundException | SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- Example of PREPARE EXECUTE in gsql

```
-- Create the reason table.
gaussdb=# CREATE TABLE reason (
    CD_DEMO_SK          int NOT NULL,
    CD_GENDER           varchar(10),
    CD_MARITAL_STATUS   varchar(10)
);

-- Create a prepared statement for an INSERT statement and execute the prepared statement.
gaussdb=# PREPARE insert_reason(int,varchar(10),varchar(10)) AS INSERT INTO reason
VALUES($1,$2,$3);
gaussdb=# EXECUTE insert_reason(52, 'AAAAAAAADD', 'reason 52');

-- Query data.
gaussdb=# SELECT * FROM reason;
 cd_demo_sk | cd_gender  | cd_marital_status
------------+------------+-------------------
         52 | AAAAAAAADD | reason 52
(1 row)

gaussdb=# SELECT name,statement,parameter_types,from_sql,is_deleted,size, valid, is_reused FROM
gs_all_prepared_statements WHERE statement LIKE '%reason%';
     name      |                                    statement                                     |
parameter_typ
es                 | from_sql | is_deleted | size  | valid | is_reused
---------------+----------------------------------------------------------------------------------------------
+------------------------------
--------------------+----------+------------+-------+-------+-----------
 insert_reason | PREPARE insert_reason(int,varchar(10),varchar(10)) AS INSERT INTO reason
VALUES($1,$2,$3); | {integer,"character varying","
character varying"} | t        | f          | 23552 | t     | f
(1 row)


-- Drop the reason table.
gaussdb=# DROP TABLE reason;
```

- Stored procedure example

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE TABLE t1(a int);
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE test_proc AS
BEGIN
    INSERT INTO t1 VALUES (1);
END;
/
CREATE PROCEDURE
gaussdb=# CALL test_proc();
 test_proc
-----------

(1 row)

-- Query a view.
gaussdb=# SELECT func_signature,query FROM
pg_catalog.gs_plsql_statements_details(pg_current_sessid());
func_signature                                |           query
----------------------------------------------------------------------------------------+--------------------------  name:
test_proc(), fn_collation: 0, search_path: addUser: false, schemas: 2200 | INSERT INTO t1 VALUES (1)
(1 row)

gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP PROCEDURE test_proc();
DROP PROCEDURE
```

7.35.1.4 FAQ

1. **Symptom**: Why is a cached plan not reused?

   Cause analysis: If the URL parameters such as **preparethreshold** are incorrectly set, the plan cannot be cached and reused.

   Solution: For details, see **Connection Parameter Reference**. The statement key is specified only when the SQL statement is sent for the *prepareThreshold*th time. The plan is cached in the session and waits for subsequent reuse. Before the *prepareThreshold*th time, the unnamed statement without the statement key is sent. The plan of such statement cannot be reused and will be cleared and overwritten each time a new SQL statement is delivered.

2. **Symptom**: Too many plan caches exist.

   Cause analysis: JDBC repeatedly creates new prepared statements and does not close the original statements. As a result, there are too many cached plans. The **preparedStatementCacheQueries** and **preparedStatementCacheSizeMiB parameters** in the URL can be used to control the number of cached queries in a session. However, the close method must have been called for the prepared statement specified by JDBC.

   Solution: Modify the JDBC statement to proactively close the prepared statements that are no longer used, or use the functionality of controlling the upper limit of the plan cache memory of GaussDB.

7.35.2 Automatic Parameterization and Plan Reuse for Identical Statements

7.35.2.1 Overview

Description

- Plan reuse for identical statements: This is a technology that optimizes query performance by caching execution plans. The core principle is to cache SQL statements and generated execution plans. For an SQL statement with the same text (including all literal values, case, and spaces) as a previously cached statement, the cached plan is directly reused, thereby skipping the repetitive parsing and optimization phases to save time. This functionality supports simple queries sent by GSQL and PBE (including batch) scenarios. The SQL text can contain user-specified bound parameters.
- Automatic parameterization: During the parameterization process, literals in SQL statements are replaced with bound parameter variables. If the parameterized SQL statements are the same, the same plan template can be reused. Simple queries and PBE scenarios without bound parameters (including batch) are supported. The original SQL statement cannot contain user-specified bound parameters. If the original SQL statement contains user-specified bound parameters, only plan reuse for identical statements is supported.

Plan reuse for identical statements and automatic parameterization can be controlled by the GUC parameter **cursor_sharing**. For details about its behavior, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.

- When the GUC parameter **cursor_sharing** is set to **force**, both automatic parameterization and plan reuse for identical statements are enabled. For statements that meet the automatic parameterization conditions, the automatic parameterization is used for caching preferentially. If the conditions are not met, the plan reuse for identical statements is selected. For example, in MySQL- or M-compatible mode, parameterization is not supported and **cursor_sharing** is automatically set to **exact**.
- When the GUC parameter **cursor_sharing** is set to **exact**, only plan reuse for identical statements is enabled.
- When the GUC parameter **cursor_sharing** is set to **off**, automatic parameterization and plan reuse for identical statements are disabled.

  If the plan template cannot be reused, the performance cannot be improved through plan reuse. If a large number of different SQL templates are stored, the plan cache memory usage is too high and reaches the upper limit, triggering the planned memory eviction. As a result, the performance deteriorates and even drops to zero in extreme scenarios.

Specifications and Limitations

- Plan reuse for identical statements:
  - Only simple query statements and PBE statements are supported.
  - Only DML statements are supported.
  - Statements in stored procedures are not supported.
  - In M-compatible mode, the SHOW INDEX statement and SELECT INTO OUTFILE statement are not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

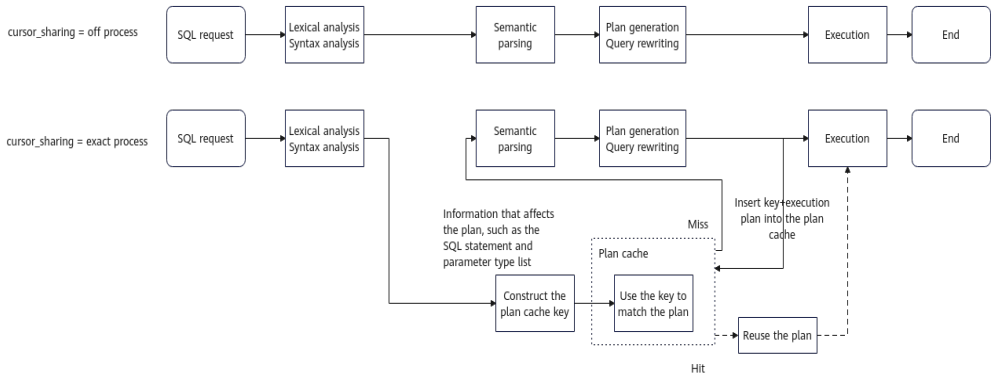
- Automatic parameterization:
  - Only simple query statements and PBE statements without bound parameters (including batch scenarios) are supported.
  - Only DML statements are supported.
  - Statements in stored procedures are not supported.
  - Statements that contain bound parameters are not supported.
  - In the light proxy scenario, statements containing GSI tables are not supported.
  - Statements that contain database link tables are not supported.
  - MySQL-compatible databases are not supported.
  - In M-compatible mode, automatic parameterization is not supported when the GUC parameter **m_format_behavior_compat_options** includes the **select_column_name** option.
  - In M-compatible mode, statements containing user variables do not support automatic parameterization.
  - In M-compatible mode, the SHOW INDEX and SELECT INTO outfile statements do not support automatic parameterization.
  - Some constants cannot be parameterized, including precision constants following types such as int(n), column constants following ORDER BY/ GROUP BY, constants in hints, constants following _charset, and constants in partitions of partitioned tables.

7.35.2.2 Technical Principle

Core Mechanism

- Plan reuse for identical statements

Figure 7-72 Principle of reusing identical statement plans



After lexical and syntax parsing is performed on a user's SQL statement, the same SQL statement plan can be reused. When a user executes an SQL statement for the first time, the database performs semantic parsing on the SQL statement and generates a plan. In addition, the database uses the semantic parsing result and plan and constructs a hash table for the plan cache based on key information such as the SQL statement as the key, and then caches the result and plan. When the database encounters an identical

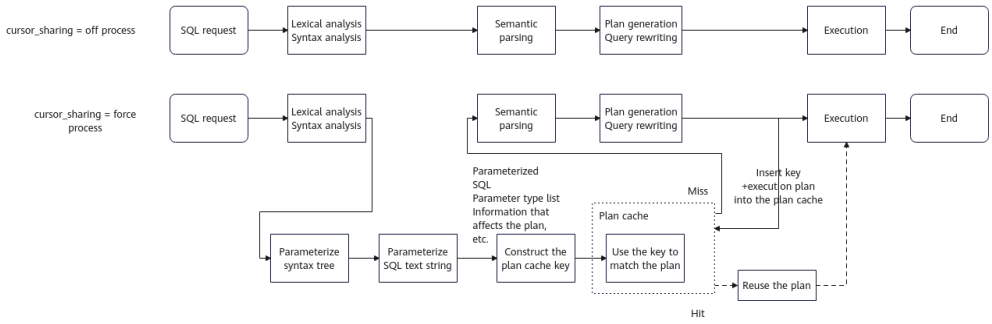
SQL statement (everything must be the same, including all literal values, cases, and spaces) again, it matches the plan cache in the hash table based on the key and directly reuses the identical cached plan. In this way, the database skips the repetitive parsing and optimization phases to save time.

Compared with plan reuse for different statements, plan reuse for identical statements adds the following operation logic:

- Constructing a plan cache key
- Using the key to match a plan
- Reusing the plan
- Storing information about the key and execution plan

- Automatic parameterization

Figure 7-73 Principle of automatic parameterization



a. After the lexical and syntax parsing is performed on a user's SQL statement to generate the original syntax tree, the automatic parameterization module traverses the syntax tree nodes and parameterizes constant nodes. The operations include converting constant nodes to parameter nodes and recording information such as parameter types.

b. An SQL template is generated based on the parameterized node information.

c. When a template SQL statement is executed for the first time, the database engine performs the following operations based on the parameterized syntax tree:

   - Semantic analysis
   - Plan generation
   - Plan caching

d. When identical SQL template statements are executed again, the cache reuse mechanism is enabled. That is, identical template SQL statements will reuse the cached plan.

Compared with the non-parameterization process, the automatic parameterization process adds the following operation logic:

- Parameterizing the syntax tree
- Parameterizing the SQL text string

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Constructing a plan cache key
- Using the key to match a plan
- Reusing the plan
- Storing information about the key and execution plan
- Executing bound parameterized constants

API Description

GUC Parameter	GUC Parameter Description
cursor_sharing	Specifies whether to enable automatic parameterization and plan reuse for identical statements. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.

Hint	Hint Description
cursor_sharing_exact	Specifies that a single SQL statement reuses the plan for identical statements. Its functionality is the same as that of **cursor_sharing** to **exact**. For details, see "Hint-based Optimization > Hint Specifying cursor_sharing to exact" in *SQL Optimization Guide*.

Function or View	Function or View Description
DBE_PERF.GS_PLANCACHE_STAT	Displays information about statements of reusable plans in all sessions in the current process. To access this view, you must be the initial user or have the MONITORADMIN permission. For details, see "Schema > DBE_PERF Schema > Query > DBE_PERF.GS_PLANCACHE_STAT" in *Reference*.
GS_ALL_PREPARED_STATEMENTS	The **is_reused** column is added to the view to indicate whether a cache is reused. For details, see "System Catalogs and System Views > System Views > Other System Views > GS_ALL_PREPARED_STATEMENTS" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function or View	Function or View Description
DBE_PERF.GLOBAL_PLAN CACHE_STATUS	The **parameterized_info** column is added to the view, indicating that a statement is executed through the **exact** or **force** process of **cursor_sharing**. For details, see "Schema > DBE_PERF Schema > Global Plan Cache > DBE_PERF.GLOBAL_PLANCACHE_STATUS" in *Reference*.
DBE_PERF.GS_DIST_PLAN CACHE_STAT	Displays information about reusable plans of all processes (including standby nodes). To access this view, you must be the initial user or have the MONITORADMIN permission. For details, see "Schema > DBE_PERF Schema > Query > DBE_PERF.GS_DIST_PLANCACHE_STAT" in *Reference*.
pg_catalog.gs_dist_planc ache_stat()	Displays the system function that contains the reusable plan statement information of all processes (including the standby node). The function returns the same information as the DBE_PERF.GS_DIST_PLANCACHE_STAT view. To view this function, you must have the initial user or MONITORADMIN permission. For details, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.
pg_catalog.gs_plancache_ stat()	Displays the reusable plan statement information of all sessions in the current process. The function returns the same information as the DBE_PERF.GS_PLANCACHE_STAT view. To view this function, you must have the initial user or MONITORADMIN permission. For details, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.
DBE_PERF.global_plancac he_status(is_with_plan)	Displays the GPC status information on all nodes, that is, the **plan_info** column. The input parameter **is_with_plan** is of the Boolean type. The default value is **false**. If the input parameter is **true**, the **plan_info** information is displayed. The function returns the same information as DBE_PERF.GLOBAL_PLANCACHE_STATUS. For details, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.

When the GUC parameter **cursor_sharing** is set to **force** and automatic parameterization is enabled, you can use EXPLAIN to check whether a plan uses automatic parameterization.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

EXPLAIN Mode	Plan Display Description
normal/pretty	The parameterized plan displays the [Parameterized] tag.

7.35.2.3 Operation Guide

- If **cursor_sharing** is set to **exact**, only the functionality of plan reuse for identical statements is enabled.

  a. Create a database and tables, and set parameters.

```
gaussdb=# CREATE DATABASE ora_db dbcompatibility = 'ORA';
CREATE DATABASE
gaussdb=# \c ora_db
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora_db" as user "yumingyue".
ora_db=# CREATE TABLE cursor_sharing_exact_tbl(col1 int, col2 int) DISTRIBUTE BY hash(col1);
CREATE TABLE
ora_db=# INSERT INTO cursor_sharing_exact_tbl VALUES(1, 1);
INSERT 0 1
ora_db=# SET cursor_sharing = exact;
SET
```

  b. Execute data query.

```
ora_db=# SELECT * FROM cursor_sharing_exact_tbl WHERE col1 = 1;
 col1 | col2
------+------
    1 |    1
(1 row)
```

  c. The dbe_perf.gs_plancache_stat view displays the cache information about the statement.

```
ora_db=# SELECT parameterized_sql, parameter_types, is_parameterized FROM
dbe_perf.gs_plancache_stat WHERE parameterized_sql LIKE '%from cursor_sharing_exact_tbl%';

parameterized_sql                                                 | paramet
er_types | is_parameterized
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+--------
---------+------------------
 select * from cursor_sharing_exact_tbl where col1 =
1;                                                                | {}
        | f
 select parameterized_sql, parameter_types, is_parameterized from dbe_perf.gs_plancache_stat
where parameterized_sql like '%from cursor_sharing_exact_tbl%'; | {}
        | f
(2 rows)
```

  d. If GPC is enabled in the environment, query the GPC view DBE_PERF.GLOBAL_PLANCACHE_STATUS. If the **parameterized_info** column is marked as **exact**, the statement is executed through plan reuse for identical statements. If GPC is not enabled in the environment, enable **enable_global_plancache**, restart the database, and perform **a** to **c** again.

```
ora_db=# SHOW enable_global_plancache;
 enable_global_plancache
-------------------------
 on
(1 row)

ora_db=# SELECT nodename, query, parameterized_info FROM
DBE_PERF.GLOBAL_PLANCACHE_STATUS WHERE query LIKE '%from cursor_sharing_exact_tbl%';
 nodename  |                                                 query
   | parameterized_info
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-----------
+-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------
---+--------------------
 cn1       | select parameterized_sql, parameter_types, is_parameterized from
dbe_perf.gs_plancache_stat where parameterized_sql like '%from cursor_sharing_exact_tbl%
'; | exact
 cn1       | select nodename, query, parameterized_info from
DBE_PERF.GLOBAL_PLANCACHE_STATUS where query like '%from cursor_sharing_exact_tbl%';
   | exact
 cn1       | select * from cursor_sharing_exact_tbl where col1 = 1;
   | exact
 datanode1 | select * from cursor_sharing_exact_tbl where col1 = 1;
   | exact
(4 rows)
```

e. Drop the database and restore the environment.

```
ora_db=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "yumingyue".
gaussdb=# DROP DATABASE ora_db;
DROP DATABASE
```

- When **cursor_sharing** is set to **force**, both automatic parameterization and plan reuse for identical statements are enabled.

  a. Create a database and tables, and set parameters.

  ```
  gaussdb=# CREATE DATABASE ora_db dbcompatibility = 'ORA';
  CREATE DATABASE
  gaussdb=# \c ora_db
  Non-SSL connection (SSL connection is recommended when requiring high-security)
  You are now connected to database "ora_db" as user "yumingyue".
  ora_db=# CREATE TABLE cursor_sharing_force_tbl(col1 int, col2 int) DISTRIBUTE BY hash(col1);
  CREATE TABLE
  ora_db=# INSERT INTO cursor_sharing_force_tbl VALUES(1, 1);
  INSERT 0 1
  ora_db=# SET cursor_sharing = force;
  SET
  ```

  b. View the plan by using EXPLAIN. The parameterized plan displays the [Parameterized] tag.

  ```
  ora_db=# EXPLAIN (costs off) SELECT * FROM cursor_sharing_force_tbl WHERE col1 = 1;
       QUERY PLAN
  ---------------------
   [Parameterized]
   Data Node Scan
     Node/s: datanode1
  (3 rows)
  ```

  c. Run the following statement:

  ```
  ora_db=# SELECT * FROM cursor_sharing_force_tbl WHERE col1 = 1;
   col1 | col2
  ------+------
      1 |    1
  (1 row)
  ```

  d. The dbe_perf.gs_plancache_stat view displays the cache information about the statement. For example, the **parameterized_sql** column displays the parameterized result of the SQL statement.

  ```
  ora_db=# SELECT parameterized_sql, parameter_types, is_parameterized FROM
  dbe_perf.gs_plancache_stat WHERE parameterized_sql LIKE '%from cursor_sharing_force_tbl%';
                      parameterized_sql                      | parameter_types | is_parameterized
  -----------------------------------------------------------+-----------------+------------------
   select * from cursor_sharing_force_tbl where col1 = ?; | {integer}       | t
  (1 row)
  ```

  e. Query the dbe_perf.statement view. The **query** column displays the parameterized statement information. The dollar sign ($) indicates the parameterized constant. (Ensure that the number of unique SQL records

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

is less than the maximum number of unique SQL records that can be recorded, which is specified by **instr_unique_sql_count**.)

```
ora_db=# SELECT query FROM dbe_perf.statement WHERE query LIKE '%from
cursor_sharing_force_tbl%';
                          query
-----------------------------------------------------------
 select * from cursor_sharing_force_tbl where col1 = $1;
(1 row)
```

f. Drop the database and restore the environment.

```
ora_db=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "yumingyue".
gaussdb=# DROP DATABASE ora_db;
DROP DATABASE
```

7.35.2.4 FAQ

1. **Question**: Are there plan change issues in automatic parameterization?

   Answer: Automatic parameterization may cause the plans of some statements to change. Automatic parameterization replaces constants in statements with parameters and then binds parameters for execution. During plan generation, constants and parameters are processed differently in some scenarios. Therefore, after **cursor_sharing** is set to **force**, some plans may change.

   In the following example, **select * from cursor_sharing_tbl where col1 = 1;** uses index scan when **cursor_sharing** is set to **off**, and uses full table scan when **cursor_sharing** is set to **force**.

```
gaussdb=# CREATE DATABASE ora_db dbcompatibility = 'ORA';
CREATE DATABASE
gaussdb=# \c ora_db
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora_db" as user "yumingyue".
ora_db=# CREATE TABLE cursor_sharing_tbl(col1 int, col2 int) DISTRIBUTE BY hash(col1);
CREATE TABLE
ora_db=# CREATE INDEX cursor_sharing_tbl_idx ON cursor_sharing_tbl(col1) WHERE col1 != 0;
CREATE INDEX
ora_db=# SET plan_cache_mode = 'force_generic_plan';
SET
ora_db=# SET max_datanode_for_plan = 1;
SET
ora_db=# SET cursor_sharing = off;
SET
ora_db=# EXPLAIN verbose SELECT * FROM cursor_sharing_tbl WHERE col1 = 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: cursor_sharing_tbl.col1, cursor_sharing_tbl.col2
   Node/s: datanode1
   Remote query: SELECT col1, col2 FROM public.cursor_sharing_tbl WHERE col1 = 1

 Remote SQL: SELECT col1, col2 FROM public.cursor_sharing_tbl WHERE col1 = 1
 Datanode Name: datanode1
   Bitmap Heap Scan on public.cursor_sharing_tbl  (cost=1.36..9.58 rows=11 width=8)
     Output: col1, col2
     Recheck Cond: (cursor_sharing_tbl.col1 = 1), (Expression Flatten Optimized)
     ->  Bitmap Index Scan on cursor_sharing_tbl_idx  (cost=0.00..1.36 rows=11 width=0)
           Index Cond: (cursor_sharing_tbl.col1 = 1)

(13 rows)

ora_db=# SET cursor_sharing = force;
SET
ora_db=# EXPLAIN verbose SELECT * FROM cursor_sharing_tbl WHERE col1 = 1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                              QUERY PLAN
-------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: cursor_sharing_tbl.col1, cursor_sharing_tbl.col2
   Node/s: datanode1
   Remote query: SELECT col1, col2 FROM public.cursor_sharing_tbl WHERE col1 = 1

 Remote SQL: SELECT col1, col2 FROM public.cursor_sharing_tbl WHERE col1 = 1
 Datanode Name: datanode1
   [Parameterized]
   Seq Scan on public.cursor_sharing_tbl  (cost=0.00..36.86 rows=11 width=8)
     Output: col1, col2
     Filter: (cursor_sharing_tbl.col1 = $1), (Expression Flatten Optimized)

(13 rows)

ora_db=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "yumingyue".
gaussdb=# DROP DATABASE ora_db;
DROP DATABASE
```

In this scenario, the constant 1 in **SELECT * FROM cursor_sharing_tbl WHERE col1 = 1;** is used as a parameter to generate a plan. When **plan_cache_mode** is set to **'force_generic_plan'** and the Gplan is forcibly used, the index conditions of some indexes in **cursor_sharing_tbl_idx** are used as parameters, and the indexes become invalid. Therefore, full table scan is performed.

In the plan change scenario, you can set **plan_cache_mode** to **'force_custom_plan'** to forcibly use the Cplan, or disable automatic parameterization.

2. **Question**: Do automatic parameterization and plan reuse for identical statements generate too many plan caches?

   Answer: Automatic parameterization and plan reuse for identical statements cache the plans of executed statements to prevent repeated plan generation in scenarios where plans can be reused, thereby accelerating query execution. However, this mechanism may cause the execution plan cache to occupy excessive memory. The system has enabled the functionality of controlling the upper limit of the plan cache. If the session-level plan cache usage is abnormal, you can adjust the value of **max_plancache_memory_ratio** parameter as needed.

7.35.3 Global Plan Cache

7.35.3.1 Overview

Description

The global plan cache (GPC) functionality enables execution plans to be shared among different sessions. GPC takes effect based on the prepared plan cache, automatic parameterization, and plan reuse for identical statements. It stores the plans cached in a session to GPC, making them globally visible. Other sessions can obtain the required plans from GPC based on the actual SQL statements and reuse the cached shared plan for execution.

GPC can be controlled by the GUC parameter **enable_global_plancache**. If GPC is disabled (that is, the GUC parameter **enable_global_plancache** is set to **off**), each

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

session caches only the plan executed by itself. That is, if the same query is performed in the same service session, a private plan needs to be generated for each session. After GPC is enabled (that is, the GUC parameter **enable_global_plancache** is set to **on**), the plan can be obtained globally. The new connection can directly obtain the plan from the shared plan cache (not generate the plan independently). Therefore, GPC can reduce the memory occupied by repeated plan caches between sessions, and enable new sessions to directly use the cached plans, reducing the plan generation overhead on the new sessions and improving the query efficiency.

NOTE

- The value of the GUC parameter **enable_global_plancache** configured on each CN or DN must be the same. Otherwise, an error will be reported.

Scenarios

When the database is running, if the execution plan is cached only in the current session, a large amount of memory is wasted. GPC can save a large amount of memory.

Specifications and Limitations

1. The specified prepared plan cache and plan cache of automatic parameterization and plan reuse for identical statements can be shared globally.
2. A plan can be stored in the shared plan cache only when the generic plan is generated.
3. The plans of temporary tables and global temporary tables are not stored in the shared plan cache even if the generic plan is generated.
4. The STREAM plan cannot be stored in the shared plan cache.
5. When the client can directly connect to a DN, the PBE or PREPARE EXECUTE statement cannot be executed.

7.35.3.2 Technical Principles

Core Mechanism

The following uses PreparedStatement as an example. **Figure 7-74** shows the memory location when the plan is cached in the current session. **Figure 7-75** shows the memory location when GPC is enabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 7-74 Memory location when GPC is disabled

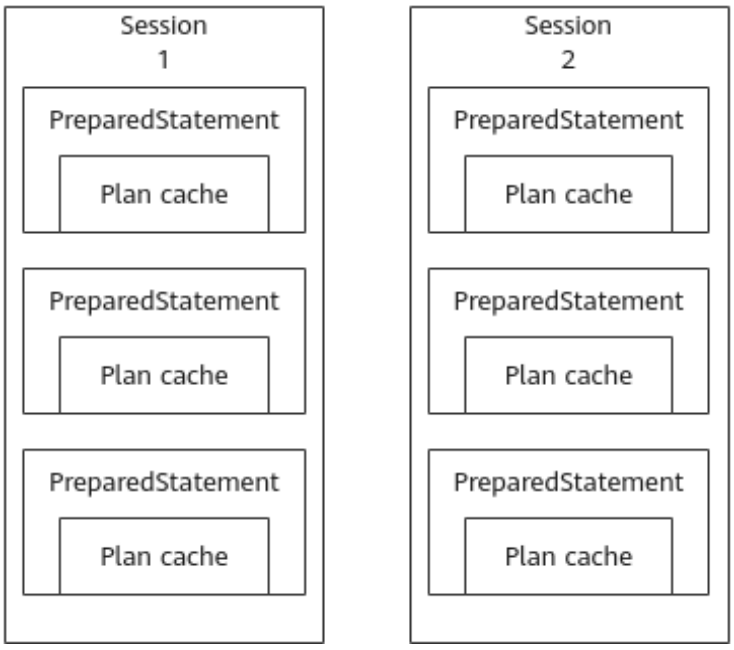
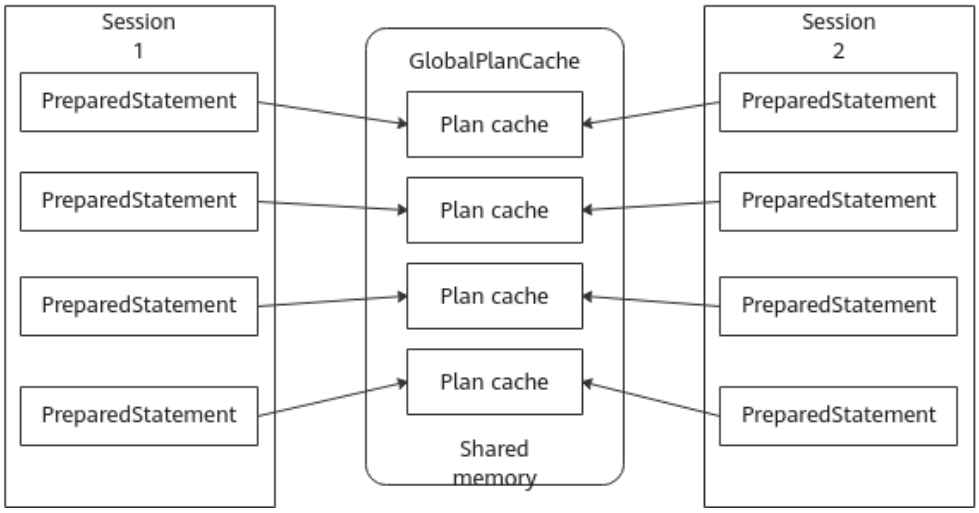


Figure 7-75 Memory location when GPC is enabled



You can specify the prepared plan cache. Prepared statements are generated in a session based on the specified key (SQL statements and session-level GUC parameters). The key is used to match the shared plan cache and directly reference the matched cached plan. Automatic parameterization, plan reuse for identical statements, and plan cache sharing in stored procedures have similar logic.

- Plans are cached in sessions. Therefore, the plan cache reuse between CNs and DNs is coupled with sessions. After a CN is connected to a DN to execute PBE, the connection is not released. A persistent connection must be retained to ensure that the expected plan cache can be obtained when the statement key is forwarded from the CN to the session on the DN. The connection to the DN can be released only after the session on the CN exits.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

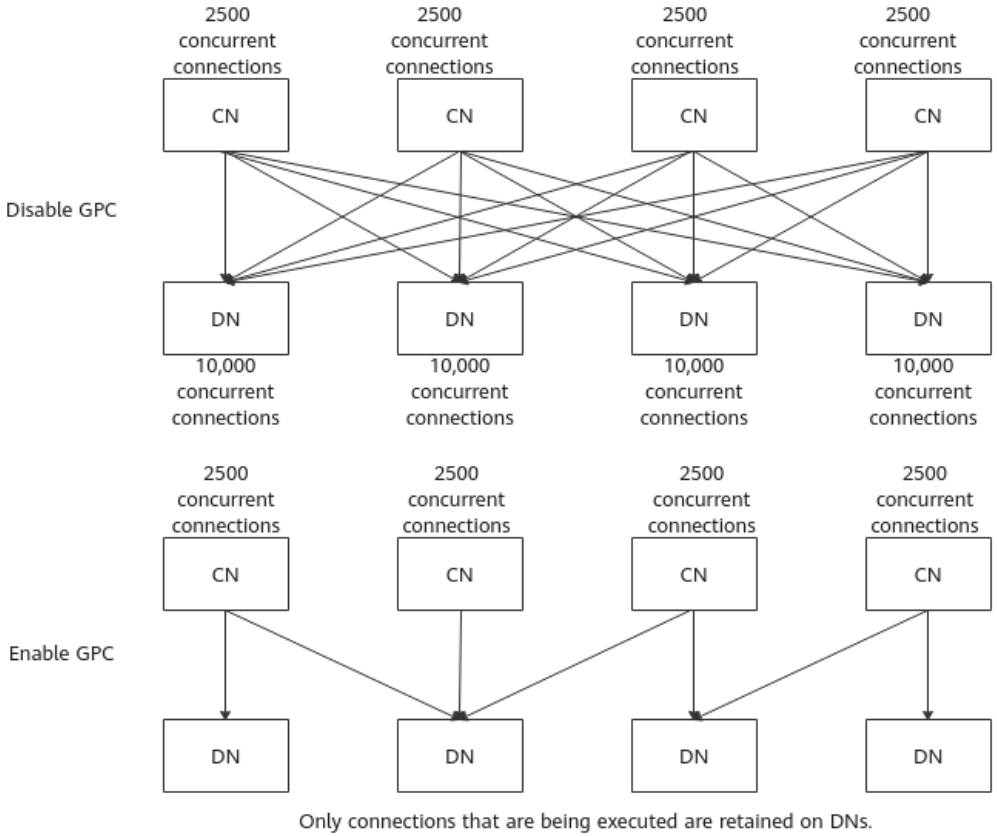
- In high-concurrency scenarios, DNs will become the connection bottleneck of the cluster.

  As shown in **Figure 7-76**, if each CN has 2500 concurrent connections, and there are four CNs and four DNs in the cluster, each DN has 10,000 concurrent connections because all PBE connections of the CNs are retained on the DNs. Each session occupies memory. Therefore, the main bottleneck of DNs is memory, and the overall concurrency capability of the system is limited by the concurrency capability of a single DN.

  When GPC is enabled, a CN directly forwards the original PBE command to the DN each time a prepared statement is executed. Different from the prepared plan cache scenario, the plan cache is not obtained from the session using the statement name as the key. Instead, the SQL text in the PBE forwarded to the DN each time is used as the key to obtain the plan cache from GPC.

  With GPC, you do not need to retain sessions for caching plans. In this way, sessions between the CN and DNs can be decoupled from prepared statements, and persistent connections do not need to be retained. Therefore, the number of connections on the DN is the number of connections that are being executed. After the execution is complete, the connections can be directly released. In addition, the DN does not cache prepared statements, and the original SQL statements are parsed directly each time.

  Figure 7-76 Comparison of the number of connections between the CN and DNs when GPC is enabled or disabled



API Description

Name		Description
GUC parameters	enable_global_plancache	Enables GPC. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
	gpc_clean_timeout	Specifies the retention period of unused shared plans. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
	disable_gpc_store	Prevents subsequent plans from being stored in GPC. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
	enable_gpc_grayrelease_mode	Enables GPC without restarting the cluster. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
Hints	no_gpc	Forcibly prevents a single query statement from being cached in GPC. For details, see "Hint-based Optimization" in *SQL Optimization Guide*.
Functions or views	DBE_PERF.GLOBAL_PLANCACHE_STATUS	Displays the GPC status. For details, see "Schema > DBE_PERF Schema > Global Plan Cache" in *Reference*.
	DBE_PERF.LOCAL_PLANCACHE_STATUS	Displays the GPC status of the current node. For details, see "Schema > DBE_PERF Schema > Global Plan Cache" in *Reference*.
	DBE_PERF.LOCAL_PLANCACHE_CEALN	Clears unused GPC on the current node. For details, see "Schema > DBE_PERF Schema > Global Plan Cache" in *Reference*.
	DBE_PERF.GLOBAL_PLANCACHE_CLEAN	Clears unused GPC on the current node. For details, see "Schema > DBE_PERF Schema > Global Plan Cache" in *Reference*.
	pg_catalog.plancache_status(is_with_plan)	Displays the GPC status information on the current node, that is, the **plan_info** column. The input parameter **is_with_plan** is of the Boolean type. The default value is **false**. If the input parameter is **true**, the **plan_info** information is displayed. For details, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name		Description
	dbe_perf.global_plancache_status(is_with_plan)	Displays the GPC status information on all nodes, that is, the **plan_info** column. The input parameter **is_with_plan** is of the Boolean type. The default value is **false**. If the input parameter is **true**, the **plan_info** information is displayed. For details, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.
	DBE_PERF.global_plancache_clean()	Clears unused GPC on all nodes. For details, see "SQL Reference > Functions and Operators > System Administration Functions > Other Functions" in *Reference*.
	pg_catalog.plancache_clean()	Clears unused GPC on the current node. For details, see "SQL Reference > Functions and Operators > System Administration Functions > Other Functions" in *Reference*.
	pg_catalog.global_plancache_set_invalid(hashkey, unique_sql_id)	Allows you to manually invalidate the cached plans in GPC on the current node by using the input parameters **hashkey** and **unique_sql_id** of the bigint type. For details, see "SQL Reference > Functions and Operators > System Administration Functions > Other Functions" in *Reference*.
	pg_catalog.global_plancache_show_plan (hashkey, unique_sql_id)	Allows you to view the cached plans in GPC of the corresponding **hashkey** and **unique_sql_id** by using the input parameters **hashkey** and **unique_sql_id** of the bigint type. For details, see "SQL Reference > Functions and Operators > System Administration Functions > Other Functions" in *Reference*.

7.35.3.3 Operation Guide

Prerequisites

- Enable cache sharing for execution plans of statements in PBE queries and stored procedures. Set **enable_global_plancache** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
- Restart the cluster for the preceding operations to take effect.
- Subsequent execution plans can be stored in the global plan cache.

```
SET disable_gpc_store = off;
```

NOTE

In this example, a distributed cluster with 1 CN and 2 DNs is used. The configuration results of other clusters may be slightly different.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Procedure

- JDBC example

  a. Prepare data.

```
gaussdb=> CREATE TABLE test_gpc_jdbc(col_a int, col_b int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col_a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=> INSERT INTO test_gpc_jdbc VALUES (generate_series(1, 1000), generate_series(1, 1000));
INSERT 0 1000
```

  b. Execute the JDBC.

```
// The following uses the gsjdbc4.jar driver as an example. If other driver packages are used, you only need to change the driver class name and the URL prefix of the database connection.
// gsjdbc4.jar: The main class name is org.postgresql.Driver, and the URL prefix of the database connection is jdbc:postgresql.

public static void main(String[] args) throws IOException {
    // In the URL string, prepareThreshold = 1 indicates that the first statement is a named PBE.
    String urls = "jdbc:postgresql://localhost:48100/postgres?prepareThreshold=1";
    // Set the username and password in the environment variables USER and PASSWORD, respectively.
    String username = System.getenv("USER");
    String passwd = System.getenv("PASSWORD");
    Properties urlProps = new Properties();
    urlProps.setProperty("user", username);
    urlProps.setProperty("password", passwd);

    try {
        Connection con = DriverManager.getConnection(urls, urlProps);
        con.setAutoCommit(true);
        Statement stmt = con.createStatement();
        // To quickly check whether the plan is stored in the cache, set plan_cache_mode to force_generic_plan.
        stmt.execute("set plan_cache_mode = 'force_generic_plan';");
        // Call the same connection API to create a prepareStatement and execute the statement.
        String selectsql = "select * from test_gpc_jdbc where col_a = ? and col_b = ?;";
        PreparedStatement pstmt = con.prepareStatement(selectsql);
        pstmt.setInt(1, 100);
        pstmt.setInt(2, 200);
        pstmt.execute();

        stmt.close();
        pstmt.close();
        con.close();
    } catch (SQLException e) {
        e.printStackTrace();
        System.exit(1);
    }
}
```

  c. Log in to the default database postgres and check the GPC view. The executed statements can be queried and cleared through the system view.

```
gaussdb=> SELECT query, refcount, valid FROM dbe_perf.global_plancache_status WHERE query LIKE '%test_gpc_jdbc%';
                            query                            | refcount | valid
-------------------------------------------------------------+----------+-------
 select * from test_gpc_jdbc where col_a = $1 and col_b = $2 |        0 | t
 select * from test_gpc_jdbc where col_a = $1 and col_b = $2 |        0 | t
(2 rows)

gaussdb=> SELECT * FROM dbe_perf.global_plancache_clean();
 global_plancache_clean
------------------------
 t
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=> SELECT query, refcount, valid FROM dbe_perf.global_plancache_status WHERE query
LIKE '%test_gpc_jdbc%';
 query | refcount | valid
-------+----------+-------
(0 rows)
```

d. Drop a table.

```
gaussdb=> DROP TABLE test_gpc_jdbc;
DROP TABLE
```

- Function example

a. Prepare data and a function.

```
gaussdb=> CREATE TABLE test_gpc_proc(col_a int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col_a' as the distribution column by
default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=> INSERT INTO test_gpc_proc VALUES(generate_series(1, 1000));
INSERT 0 1000
gaussdb=> CREATE OR REPLACE FUNCTION count_beyond
gaussdb-> (
gaussdb(> threshold int
gaussdb(> )
gaussdb-> RETURNS int
gaussdb-> AS $$
gaussdb$> DECLARE
gaussdb$> count_number int;
gaussdb$> BEGIN
gaussdb$>     select count(*) from test_gpc_proc where col_a > threshold into count_number;
gaussdb$>     return count_number;
gaussdb$> END;
gaussdb$> $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
gaussdb=> SET plan_cache_mode='force_generic_plan';
SET
gaussdb=> SET enable_stream_operator=off;
SET
```

b. Call a function to display the GPC view.

```
gaussdb=> SELECT count_beyond(200);
 count_beyond
--------------
          800
(1 row)

gaussdb=> SELECT query, refcount, valid FROM dbe_perf.global_plancache_status WHERE query
LIKE '%test_gpc_proc%';
                            query                            | refcount | valid
-------------------------------------------------------------+----------+-------
 select count(*) from test_gpc_proc where col_a > threshold |        1 | t
(1 row)
```

c. Drop the function and data.

```
gaussdb=> DROP FUNCTION count_beyond;
DROP FUNCTION
gaussdb=> DROP TABLE test_gpc_proc;
DROP TABLE
```

7.35.3.4 FAQ

1. Why cannot a plan be stored in the shared cache?
   - If the plan is not a generic plan or the SQL statement contains temporary tables or global temporary tables, the plan cannot be shared. For details about the restrictions, see **Specifications and Limitations**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If the shared plan cache reaches the memory upper limit and cannot be evicted, the plan cannot be stored in the shared cache. For details, see **Upper Limit of the Plan Cache Memory Control**.

2. Does the GUC parameter **enable_global_plancache** take effect only after the cluster is restarted?

   No. You can set the **enable_gpc_grayrelease_mode** parameter to enable GPC by performing a rolling restart on each node. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.

7.35.4 Upper Limit of the Plan Cache Memory Control

7.35.4.1 Overview

Description

In the database system, the session-level plan cache control and GPC mechanisms may face memory overflow risks in the PBE scenario and when the cursor_sharing functionality is enabled. Specifically, if the client frequently commits differentiated SQL statements, the kernel continuously generates new plan cache entries. In the absence of an effective cleanup mechanism, these cache entries will accumulate in the memory without any limit, which may eventually cause severe faults such as memory resource exhaustion and system unavailability.

To address this risk, GaussDB innovatively implements a dual-dimensional memory control system.

1. Session-level control: A dynamic threshold is used to limit the percentage of private plan cache memory used by a single session.
2. GPC control: The LRU-K algorithm is used to implement access control on the global shared cache, and high-quality plans that are frequently executed are preferentially retained.

This mechanism ensures that the plan cache always runs within the preset secure memory range through dual restrictions (session memory upper limit+global eviction policy). When the total cache size reaches the warning threshold, the system automatically triggers an asynchronous cleanup thread to evict low-efficiency cache entries based on the policy. This ensures query performance and effectively prevents memory overflow.

Scenarios

When the database is running, if dynamic memory resources are exhausted due to continuous accumulation of plan cache entries and the lack of an effective cleanup mechanism, severe faults such as service request execution failures and database service interruption may occur. With the memory control mechanism of this feature, the system can implement precise memory quota management for the plan cache, ensuring that key services always obtain necessary memory resources. This effectively maintains the availability of core services and significantly enhances system robustness.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifications and Limitations

1. During the process of rebuilding a plan triggered by plan cache eviction, the system retains the current GUC parameter settings and does not roll back or reset the historical parameter settings. If the GUC parameter settings are changed before the rebuild operation is performed, the newly generated execution plan is constructed based on the latest parameter settings. This mechanism may cause differences in the execution policy or resource allocation between the plan before and after the eviction. This behavior is an expected feature of the system design, not a functional exception.
2. For session-level plan cache control, only the memory occupied by the execution plan generated by the query framework declared by the PREPARED statement and the plan caching triggered by the **cursor_sharing** parameter is controlled. Other memory (such as the temporary tablespace memory allocation, session-level sorting area/hash area, user connection context memory, and other non-plan cache-related memory objects) is not controlled.
3. For GPC, full-dimensional memory control is implemented. The memory allocation of all cache plans included in GPC can be managed, and the memory of multi-dimensional plan types can be controlled, including the prepared execution plan created by the PREPARED statement and the general query plan generated by the GUC parameter **cursor_sharing**.

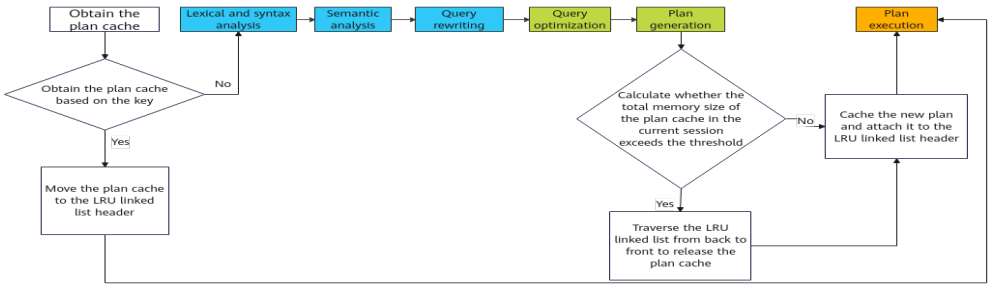
7.35.4.2 Technical Principles

Core Mechanism

- Session-level plan cache control

  After receiving an SQL statement, GaussDB performs operations such as parsing the lexicon and syntax of the SQL statement and rewriting the query. Then, it generates an execution plan, caches the plan in the session, and calculates the memory occupied by the plan. In addition, the context information required for generating the plan is saved, which is used for rebuilding the plan in case it triggers the eviction mechanism. **Figure 7-77** shows the specific process.

  Figure 7-77 Session-level plan cache control process

  The session-level plan cache control is implemented by using an LRU linked list to preferentially display the execution plans based on the execution frequency. After an execution plan is cached, the plan pointer is recorded at the header of the LRU linked list. Each time the execution plan is used, the plan is moved to the header of the linked list. That is, the execution plan closer to the head of the LRU linked list is the one that is used more frequently.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

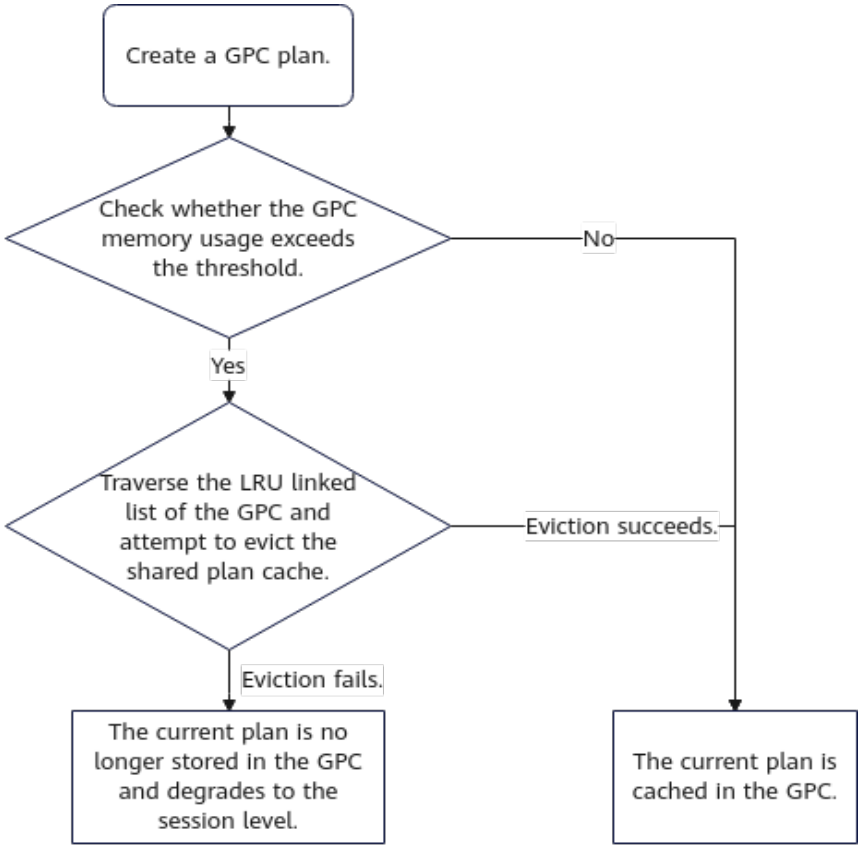
When a plan is created, the system checks whether the total size of the plan cache in the current session exceeds the specified threshold. If so, the system traverses the plan from the end of the LRU linked list and releases the plan until the total size of the session-level plan cache is less than the specified threshold.

During next execution, the system attempts to obtain the plan cache. If the plan cache already exists, the plan pointer is moved to the header of the LRU linked list and the execution continues. Otherwise, a plan needs to be created again. The process of creating a plan is the same as that of generating a plan for the first time.

- Global plan cache (GPC) control

  The GPC control is implemented based on the session-level plan cache control. A global plan may be used by multiple sessions at the same time. Therefore, the plan cannot be directly evicted and cleared to release available memory. That is, the GPC can be evicted only when no session is used. **Figure 7-78** shows the specific process.

  Figure 7-78 GPC control process



To ensure that GPC takes effect in real time, if the memory of the GPC is insufficient and cannot be evicted, a plan cannot be stored in GPC, and the session-level plan cache control is performed. The session-level plan cache is used in serial mode. In this way, the plan cache can be directly deleted when it needs to be cleared in the current session.

The system checks whether the total memory size of GPC exceeds the threshold after the current plan is stored in GPC. If the total memory size

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

does not exceed the threshold, the plan is stored in GPC. Otherwise, the system traverses the LRU linked list and attempts to evict the plan. If the eviction is successful, the plan is stored in GPC. If the eviction fails, the plan is stored only in the current session and cannot be stored in GPC.

NOTE

Rebuilding a plan after the plan cache is evicted will cause the lexical and syntax parsing process to be executed again, which brings additional performance overhead.

API Description

- GUC parameters

Table 7-62 GUC parameters for plan cache control

Control Level	Parameter	Description
Session	max_plancache_memory_ratio	This parameter is used for session-level plan cache control. For details, see the description of **max_plancache_memory_ratio** in "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.
Global shared	global_plancache_memory_ratio	This parameter is used for GPC control. For details, see the description of **global_plancache_memory_ratio** in "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

- Description of involved columns in the view

Table 7-63 Views related to plan cache control

View	Description of Involved Columns
GS_ALL_PREPARED_STATEMENTS	**size**: indicates the size of the memory occupied by the plan of the PREPARE statement, in bytes. **is_deleted**: specifies whether the plan is evicted.
DBE_PERF.GS_PLANCACHE_STAT	**size**: indicates the size of the memory occupied by the plan of the PREPARE statement, in bytes. **is_deleted**: specifies whether the plan is evicted.
DBE_PERF.GLOBAL_PLANCACHE_STATUS	**size**: indicates the size of the memory occupied by the GPC plan, in bytes.
DBE_PERF.LOCAL_PLANCACHE_STATUS	**size**: indicates the size of the memory occupied by the GPC plan, in bytes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.35.4.3 Operation Guide

7.35.4.3.1 Session-Level Plan Cache Control

Prerequisites

- Enable the logical memory management module and set **enable_memory_limit** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. After the setting is successful, restart the database cluster.
- Enable session-level plan cache control.

  Set the GUC parameter **max_plancache_memory_ratio** to a value greater than 0.

Syntax

Create a PREPARE statement using gsql or JDBC.

- Using gsql:
  ```
  PREPARE name [ ( data_type [, ...] ) ] AS statement;
  ```
- Using JDBC:
  ```
  PreparedStatement pstmt = connection.prepareStatement(sql);
  ```

Precautions

1. Session-level plan cache control only applies to the plan caches created by the PREPARE statements and automatic parameterization, but not the plan caches created by stored procedures.
2. The cache eviction policy is triggered only when the memory size reaches or exceeds the upper limit after the execution plan is created or rebuilt.

Procedure

1. Enable the logical memory management module and set the session-level plan cache control. Set **enable_memory_limit** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. After the setting is successful, restart the database cluster.
   ```
   SET max_plancache_memory_ratio = 60;
   ```
2. Create a large number of PREPARE statements.
3. Check the memory view. After the plan cache size reaches the threshold, the size does not increase. In the PREPARE statement view, there are scenarios where plans are evicted (that is, the value of **is_deleted** in the GS_ALL_PREPARED_STATEMENTS view is **t**).

Examples

```
gaussdb=# CREATE TABLE plancache_memctl_tbl (col1 int, col2 varchar(10));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO plancache_memctl_tbl (col1, col2) VALUES (1, 'test');
INSERT 0 1
gaussdb=# SET max_plancache_memory_ratio=60;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
SET
gaussdb=# SET plan_cache_mode=force_generic_plan;
SET
gaussdb=# PREPARE p1 AS SELECT * FROM plancache_memctl_tbl WHERE col1=$1;execute p1(1);
PREPARE
 col1 | col2
------+------
    1 | test
(1 row)

gaussdb=# PREPARE p2 AS SELECT * FROM plancache_memctl_tbl WHERE col1=$1;execute p2(1);
PREPARE
 col1 | col2
------+------
    1 | test
(1 row)

-- Create a large number of plan caches (based on the actual service requirements. No example will be
displayed here).

-- Query the memory view.
gaussdb=# SELECT contextname, sum(totalsize)/1024/1024 sum, count(*) count FROM
pv_session_memory_detail WHERE contextname ILIKE '%cachedplan%' GROUP BY contextname;
   contextname    |          sum          | count
------------------+-----------------------+-------
 CachedPlan       | .08398437500000000000 |     6
 CachedPlanQuery  | .02539062500000000000 |     6
 CachedPlanSource | .03515625000000000000 |     6
(3 rows)

-- Query the PREPARE statement.
gaussdb=# SELECT name,statement,size,is_deleted,valid FROM GS_ALL_PREPARED_STATEMENTS WHERE
sessionid = pg_current_sessid() ORDER BY name;
 name |                          statement                          | size  | is_deleted | valid
------+-------------------------------------------------------------+-------+------------+-------
 p1   | prepare p1 as select * from plancache_memctl_tbl where col1=$1; |     0 | t          | f
 p2   | prepare p2 as select * from plancache_memctl_tbl where col1=$1; |     0 | t          | f
 p3   | prepare p3 as select * from plancache_memctl_tbl where col1=$1; |     0 | t          | f
 p4   | prepare p4 as select * from plancache_memctl_tbl where col1=$1; |     0 | t          | f
 p5   | prepare p5 as select * from plancache_memctl_tbl where col1=$1; | 15360 | f          | t
(5 rows)

gaussdb=# DROP TABLE plancache_memctl_tbl;
DROP TABLE
```

7.35.4.3.2 GPC Control

Prerequisites

- Enable cache sharing for execution plans of statements in PBE queries and stored procedures by setting **enable_global_plancache** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in *Reference*.
- Enable the logical memory management module and set **enable_memory_limit** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. After the setting is successful, restart the database cluster.
- Enable GPC memory control.

  Set the GUC parameter **global_plancache_memory_ratio** to a value greater than 0. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

Create a PREPARE statement using gsql or JDBC.

- Using gsql:
  ```
  PREPARE name [ ( data_type [, ...] ) ] AS statement;
  ```
- Using JDBC:
  ```
  PreparedStatement pstmt = connection.prepareStatement(sql);
  ```

Precautions

1. GPC is accessed concurrently. Therefore, the eviction policy may fail.
2. If the upper limit is reached and the GPC plan fails to be evicted, the plan created again will be degraded to the session level.

Procedure

1. Enable the logical memory management and the session-level plan cache control. Set **enable_memory_limit** to **on**. For details, see "Database GUC Parameters > GUC Parameters > Resource Consumption > Memory" in *Reference*. After the setting is successful, restart the database cluster.
2. Set the **global_plancache_memory_ratio** parameter to **30**. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.
3. Create a large number of PREPARE statements.
4. Check the memory view. The plan cache size does not increase after reaching the threshold.

Examples

Set the **global_plancache_memory_ratio** parameter to **30**. For details, see "Database GUC Parameters > GUC Parameters > Workload Management" in *Reference*.

Connect to the database using gsql.

```
gaussdb=# CREATE TABLE plancache_memctl_tbl (col1 int, col2 varchar(10));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'col1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO plancache_memctl_tbl (col1, col2) VALUES (1, 'test');
INSERT 0 1
gaussdb=# SET plan_cache_mode=force_generic_plan;
SET
gaussdb=# PREPARE p1 AS SELECT * FROM plancache_memctl_tbl WHERE col1=$1;execute p1(1);
PREPARE
 col1 | col2
------+------
    1 | test
(1 row)

gaussdb=# PREPARE p2 AS SELECT * FROM plancache_memctl_tbl WHERE col1=$1;execute p2(1);
PREPARE
 col1 | col2
------+------
    1 | test
(1 row)

-- Create a large number of plan caches (based on the actual service requirements. No example will be
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
displayed here).

-- GPC plan memory size.
gaussdb=# SELECT sum(size) FROM DBE_PERF.GLOBAL_PLANCACHE_STATUS;
  sum
--------
 159744
(1 row)

gaussdb=# DROP TABLE plancache_memctl_tbl ;
DROP TABLE
```

7.35.4.4 FAQ

1. **Symptom**: After the plan cache control functionality is enabled and service statements are executed, the memory is still unavailable.

   Solution: Perform the following operations:

   a. Check whether the values of **max_plancache_memory_ratio** and **global_plancache_memory_ratio** are **0**. The value **0** indicates that the functionality is disabled.

   b. Check whether the value of **enable_memory_limit** is **off**. This parameter may be set to **off** when affected by **max_process_memory**, **shared_buffers**, and metadata. If it is set to **off**, check whether **max_process_memory** and **shared_buffers** are set properly. The plan cache control functionality takes effect only when **enable_memory_limit** is set to **on**.

   c. Decrease the values of **max_plancache_memory_ratio** and **global_plancache_memory_ratio** to reduce the cache threshold and reserve more memory resources.

2. **Symptom**: After the plan cache control functionality is enabled and service statements are executed, a plan change is detected when the cache is reused.

   Solution: Check whether there are parameter changes that affect plan generation during service execution. Due to the eviction and rebuild mechanism of the plan cache, if there are parameter changes that affect plan generation during the process, the plan before eviction may be different from that after rebuild.

7.36 DATABASE LINK

7.36.1 Overview

Description

Database links enable cross-database access. You can use a database link in the local database to connect to and access a remote database.

Scenarios

You need to access remote database tables to query and modify data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifications and Limitations

For details, see **Specifications and Limitations**.

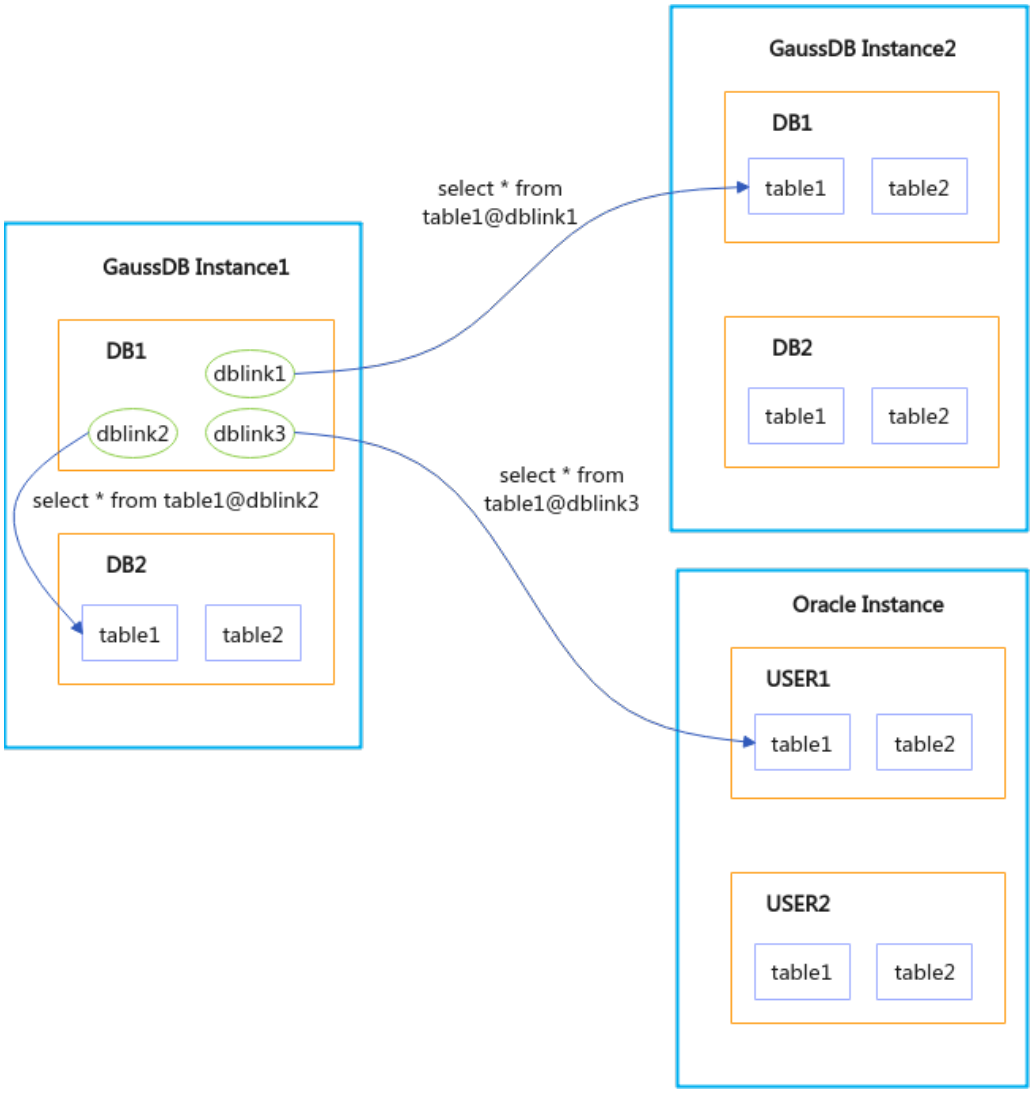
Technical Architecture

Database links can be created to access remote database objects using the "object name@dblink" syntax. This mechanism supports:

- Perform operations such as SELECT, UPDATE, and INSERT on remote tables.
- Map remote objects to local logical objects and perform operations using standard SQL syntax.
- Combine remote objects with local objects to implement cross-database object join operations.

Figure 7-79 shows how to access remote databases through database links. You can access objects in different databases of the current local instance or objects in different instances.

Figure 7-79 Accessing remote databases through database links



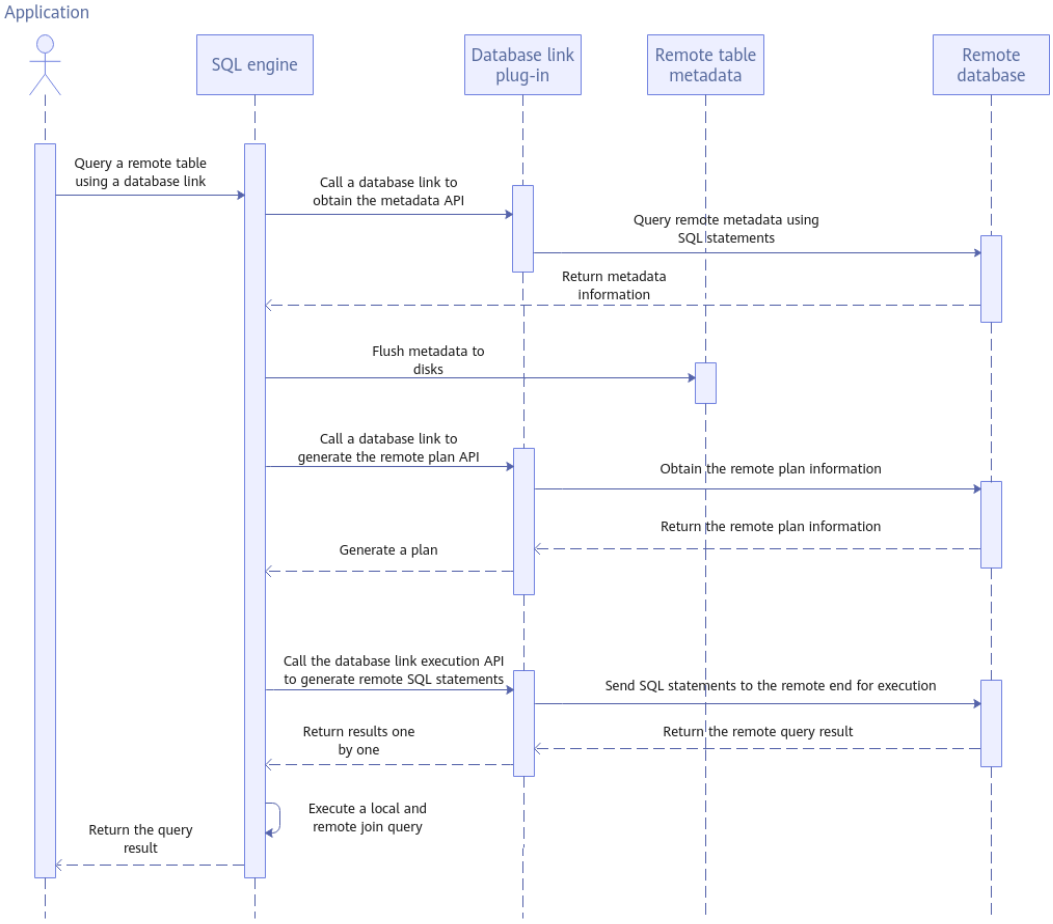
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.36.2 Technical Principle

Working Principle

Figure 7-80 shows the sequence diagram of the basic process for querying a remote table using a database link.

Figure 7-80 Sequence diagram of the basic process for querying a remote table using a database link



- Syntax parsing phase

  When a user uses a database link to query a remote object, the parser identifies the @dblink syntax identifier, generates the corresponding database link relationship node, and marks the remote object attribute.

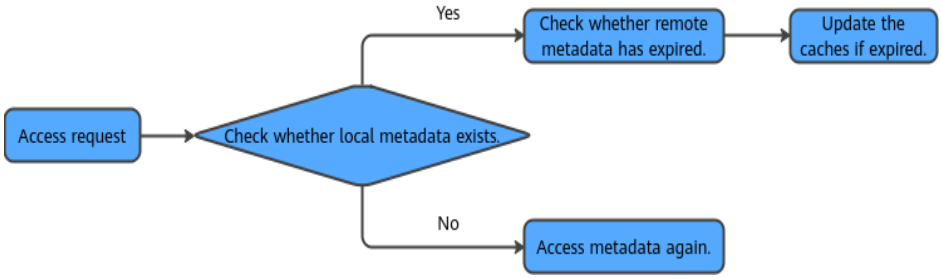
- Semantic analysis phase

  If the target table is a remote object (table@dblink):

  – The database calls the database link plug-in API to send a request for accessing metadata to the remote database and proactively starts a remote transaction.

  – The remote metadata is obtained and cached to the local database system catalogs.

  – Subsequent access process:



- Query optimization phase

  The optimizer uses the database link plug-in API to generate SQL statements to be sent to the remote database to obtain the query cost and execution plan.

- Executor phase

  The execution operator calls the database link plug-in API to send SQL requests to the remote database, obtains the remote result set, serializes it into the local data format, and passes the result set upward, layer by layer.

- Transaction commit phase

  The remote transaction status is automatically synchronized when the local transaction is committed.

API Description

Table 7-64 lists the system views used by database links. For details, see "System Catalogs and System Views > System Views > Database Links" in *Reference*.

Table 7-64 System views

View	Description
GS_DB_LINKS	Displays information about database links. You can view information about database links of your own and at the PUBLIC level.
V$DBLINK	The data in this view is obtained from GS_DB_LINKS. This view records only the information about the database links that have been used or are being used in the current session. By default, only administrators can view this view.

Table 7-65 lists the system functions used by database links. For details, see "SQL Reference > Functions and Operators > Database Link Functions" in *Reference*.

Table 7-65 System functions

Function	Description
close_database_link(dblink_name name)	Closes a database link in the current session based on the database link name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Description
gs_ora_diag(dblink_name name DEFAULT NULL::name)	Obtains information about database links connected to Oracle Database. The following information can be obtained based on database link names: gs_ora_fdw version, GaussDB version, Oracle Database's client version, and Oracle Database's server version. If the parameter is set to **NULL** and the query user is an administrator, information such as ORACLE_HOME, ORACLE_SID, TNS_ADMIN, TWO_TASK, and LDAP_ADMIN is also returned.
gs_ora_get_handle_alloc(dblink_oid oid)	Obtains the size of the memory used by the database link through the OCI based on the OID of the database link. The unit is byte.
gs_ora_get_handle_alloc_all()	Obtains the environment handle of each node and calculates the total memory size used by all database links through OCI, in bytes.
gs_ora_connection_status(dblink_oid oid)	Obtains the connection status of a database link in the current session based on the OID of the database link.
is_dblink_in_use(dblink_oid oid)	Checks whether a database link is used based on its OID.
dblink_has_updatasent(dblink_oid oid)	Checks whether a DML statement has been sent through the database link corresponding to an OID in the current transaction but has not been committed yet.
is_dblink_in_transaction(dblink_oid oid)	Checks whether a database link is in a transaction based on the OID of the database link.

Table 7-66 provides the system catalog used by database links. For details, see "System Catalogs and System Views > System Catalogs > Database Links" in *Reference*.

Table 7-66 System catalog

System Catalog	Description
GS_DATABASE_LINK	Stores database link information, specifically details about database links. Only users with the SYSADMIN permission can read this system catalog.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.36.3 Operation Guide

Preparations

- To connect to a GaussDB database, you need to enable access through database links on the remote database node and configure the remote connection. In some cases, the IP address of the DN also needs to be added to the cluster whitelist.
- Create a database link.
  - Create a database link. For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE DATABASE LINK" in *Reference*.
  - When creating a database link, you can set it to PUBLIC or PRIVATE. Private database links can be accessed only by their creators, and public database links can be accessed by all users. All information about the created database link is stored in the system view GS_DB_LINKS of the local database.
- Modify database link information.

  Modify a specified database link. For details about the syntax, see "SQL Reference > SQL Syntax > A > ALTER DATABASE LINK" in *Reference*.
- Drop a specified database link.

  Drop a specified database link. For details about the syntax, see "SQL Reference > SQL Syntax > D > DROP DATABASE LINK" in *Reference*.
- Perform the SELECT operation through a database link.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
 SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
 { * | {expression [ [ AS ] output_name ]} [, ...] }
 [ FROM from_item [, ...] ]
 [ WHERE condition ]
 [ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
 [ GROUP BY grouping_element [, ...] ]
 [ HAVING condition [, ...] ]
 [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
 [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
 [ LIMIT { [offset,] count | ALL } ]
 [ OFFSET start [ ROW | ROWS ] ]
 [ {FOR { UPDATE | SHARE  } [ OF table_name [, ...] ] } [...] ];
 {[ ONLY ] table_name [ * ] @ dblink [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
  |with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  | [function_name] ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
  | [function_name] ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
  |from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]};
```

- Perform the INSERT operation through a database link.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
  INSERT [/*+ plan_hint */] INTO table_name @ dblink [ ( column_name [, ...] ) ]
     { DEFAULT VALUES
     | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
     | query }
     [ RETURNING { {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

- Perform the UPDATE operation through a database link.

```
UPDATE [/*+ plan_hint */] [ ONLY ] table_name @ dblink [ [ AS ] alias ]
 SET {column_name = { expression | DEFAULT }
    |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
    [ FROM from_list] [ WHERE condition ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
    [ LIMIT { [offset,] count | ALL } ]
```

```
    [ RETURNING { {output_expression [ [ AS ] output_name ]} [, ...] }];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ];
```

- Perform the DELETE operation through a database link.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] FROM [ ONLY ] table_name @ dblink [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
    [ LIMIT { [offset,] count | ALL } ]
    [ RETURNING { { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

- Perform the LOCK TABLE operation through a database link.

```
LOCK [ TABLE ] {[ ONLY ] name @ dblink  [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE |
SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

- Call a stored procedure or function of the remote database.

```
CALL | SELECT [ schema. ] { func_name@dblink | procedure_name@dblink } ( param_expr );
```

NOTE

- The syntax for using a created database link to access a remote database object is similar to that for accessing a local object. The difference is that you need to add @dblink to the end of the name of the remote database object to be accessed. For details about the limitations on SQL statements, see **Table 7-68**.
- The meanings of the parameters irrelevant to database link in the preceding SQL statements are the same as those in the original SQL statements.
- To perform a join query involving a remote table and specify a column name, add @dblink to the end of the column name, indicating that a column in the remote table to which the database link points. The remote table column does not support the * notation, for example, remote_t1.*@dblink.

Specifications and Limitations

- Compatibility limitations
  - The database link feature can be used only in ORA-compatible databases.
  - The database link feature applies only to GaussDB.
  - When connecting to GaussDB through a database link, ensure that the values of the compatibility parameter **DBCOMPATIBILITY** and GUC parameters **behavior_compat_options**, **a_format_dev_version**, and **a_format_version** of the local database are the same as those of the remote database.
  - When GaussDB is connected, the following GUC parameters are automatically set before a database link starts a remote connection transaction:

```
SET search_path=pg_catalog, '$user', 'public';
SET explain_perf_mode='normal';
SET session_timeout=0;
SET datestyle=ISO;
SET intervalstyle=postgres;
SET extra_float_digits=3;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> NOTE
>
> Other parameters are set at the remote database end. If the remote parameters are different from the local parameters, the data display formats may be different. Therefore, ensure that the remote parameters are the same as the local parameters.

  - When connecting to GaussDB, ensure that the remote and local databases have the same datea type. Otherwise, the GUC parameter **mapping_date_to_datea** cannot be used.

- Character set limitations

  When a GaussDB connection is established, mismatched character sets between the local and remote databases may cause a conversion failure and trigger a remote error message. If the character encoding format of the local database is GB18030_2022, the request sent to the remote database is converted to GB18030. Therefore, if the character set of the local database is GB18030_2022, the character set of the remote database can only be GB18030 or GB18030_2022.

- Permission limitations
  - Do not use database links to connect to the initial user. The initial user is not allowed to create, modify, or drop database links.
  - The permission to create database links needs to be granted using the GRANT syntax. By default, a new user does not have the permission, but a system administrator has the permission. For details, see "SQL Reference > SQL Syntax > G > GRANT" in *Reference*.
  - When a user is granted the permission to create database links, the user is allowed to access a remote database using its IP address by default. If you do not want this to happen, do not use GRANT to grant permissions to users.
  - Permissions of local users to use database links

    If the keyword PUBLIC is specified, the database link is public and can be used by all users or schemas.

    If the keyword PUBLIC is not specified, the database link is private and can be used only by the current user or schema. (Users with the SYSADMIN permission cannot use the database link across schemas.)
  - Permissions to access remote database objects through database links

    Access to the remote database objects is controlled by user permissions of the remote database bound to the database link.

- Connection limitations
  - If the CURRENT_USER or CONNECT TO connection string is not specified, the initial username and empty password of the current database are used for connection, and the connection fails.
  - In non-multi-primary mode, you can use the local IP address and port to connect to the current database instance. CNs and DNs may be deployed on different servers. Therefore, you are advised not to set the **host** parameter to **127.0.0.1** or **localhost**. Otherwise, the connection may fail.
  - When a database link is created, the system does not check whether the link can be successfully connected. If related keywords are missing, an error may be reported when the link is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - If a key file is regenerated after a database link is created, an error is reported when the database link is used. This is because the key file used for creating the database link is different from that used for decryption.
  - In **OPTION**, **conn_opt** does not support password setting, and the input libpq parameters cannot be the same as the original ones.
- Upgrade limitations

  If the upgrade is not committed, database links cannot be created.
- Limitations on metadata and locks
  - When a database link is used to perform operations on a remote table, a schema corresponding to the remote table is created locally. If the metadata of the table does not exist locally, the metadata is written to the local system catalog. In this case, a level-7 lock is used to ensure write consistency until the transaction ends. When the database link is deleted, the corresponding metadata is also deleted.
  - When a database link is used, the table created locally is only used to store the metadata of the remote table. The table definition cannot be queried using **\d** or the pg_get_tabledef function.
  - If a long transaction uses a database link to operate a remote object for the first time, the lock is held until the transaction ends. Other transactions that use the database link for the first time are blocked. You can perform a simple query on the remote object to cache its metadata locally. In addition, if the definition of a remote table changes, the stored metadata needs to be updated locally. As a result, similar blocking occurs.
  - When you create a schema locally for a remote schema, the local schema name can contain a maximum of 63 characters in the format "*[username#]remote schema@dblink_name*". If the database link is private, *username#* cannot be omitted.
  - Some predicates cannot be pushed down. When the UPDATE, DELETE, and FOR UPDATE functions are used through database links, the lock scope may be expanded. You can query the pushdown statements from the printing plans to determine the lock scope.
  - When a database link is used to perform operations on a remote table, a single-node group is created and randomly bound to a DN.
- Transaction limitations

  When database links are used, the relationship between local and remote transactions is as follows:
  - Local transactions synchronously control the commit/rollback status of remote transactions.
  - The mapping between isolation levels is described in **Table 7-67**.

    Table 7-67 Mapping of GaussDB database isolation levels in non-multi-primary mode

Local Isolation Level	Remote Isolation Level
Read Uncommitted	Repeatable Read
Read Committed	Repeatable Read

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Local Isolation Level	Remote Isolation Level
Repeatable Read	Repeatable Read
Serializable	Serializable

NOTE

- If you commit a local transaction, a transaction commit request is sent to the remote end. If the local transaction fails to be committed due to an exception (such as a connection exception or a local cluster instance exception) after the remote transaction is successfully committed, the committed remote transaction cannot be withdrawn. As a result, the local transaction may be inconsistent with the remote transaction.
- Database links do not support the XA protocol, which may lead to inconsistency between the local and remote transactions.

- Limitations on supported SQL statements
  - For details about the statements supported by database links, see **Table 7-68**.
  - For details about the table types supported by database links, see **Table 7-69**.
- Limitations on calling database link functions and stored procedures
  - Remote functions or stored procedures can be called through database links only in non-multi-primary mode.
  - When a function in the remote database is called through the database link, the local database must create a function whose name and parameter list are the same as those of the remote function. If the remote function is overloaded, you need to create all overloaded functions in the local database. Otherwise, parameter matching fails.
  - When the stored procedures and functions in a remote database are called using a database link, the parameter or return value type cannot contain user-defined types. The **OUT**/**INOUT** parameters or parameters with a default value are not supported. In addition, package functions, aggregate functions, window functions, and set returning functions are not supported. Remote functions cannot be called using **SELECT * FROM func@dblink();**.
  - When the stored procedures or functions in a remote database are called using a database link in the PLSQL_BODY, the parameter or return value type cannot contain user-defined types. The **OUT**/**INOUT** parameters or parameters with a default value are not supported. In addition, package functions, overloaded functions, aggregate functions, window functions, and set returning functions are not supported.
  - If no schema is specified when calling stored procedures and functions in a remote database using a database link, the functions or stored procedures under PUBLIC are called by default.
  - When stored procedures and functions in a remote database are called using a database link, the parameter list does not support the := or => mode for assigning values to parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- When calling a stored procedure or function from a remote database within a PLSQL_BODY, use the following syntax: [CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } (param_expr).
- When calling a parameterless stored procedure or function from a remote database within a PLSQL_BODY, use the following syntax: [CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } ( ).

- Synonym
  - A database link name cannot be used to create a synonym.
  - When creating a synonym for a remote object accessed using a database link, if no schema is specified for the remote object, the current schema where the synonym is created is used by default.
  - Synonyms that point to a database link in a remote database cannot be called through database links. For example:
    i. Create table a (**table1**) in database a (**db1**).
    ii. Create a database link (**dblink1**) for connecting to database a (**db1**) in database b (**db2**), and create a synonym (**CREATE SYNONYM table11 FOR user1.table1@dblink1";**).
    iii. Create a database link (**dblink2**) for connecting to database b (**db2**) in database c (**db3**), and call the synonym **table11** in database b (**db2**) through **dblink2**. (If you run **SELECT * FROM table11 @dblink2;**, the error message "If the remote synonym is a dblink object, dblink access is not supported" is displayed.)
  - For details about the precautions for creating a synonym, refer to "SQL Reference > SQL Syntax > C > CREATE SYNONYM" in *Reference*.
- Table type limitations

  The table type limitations for connecting to GaussDB are as follows:
  - HASHBUCKET: The query or DML operation cannot be performed on remote hash bucket tables through database links.
  - SLICE: The query or DML operation cannot be performed on the remote slice table through DATABASE LINK.
  - Replication table: The query or DML operation cannot be performed on remote replication tables through database links.
  - TEMPORARY: The query or DML operation cannot be performed on remote temporary tables through database links.
- View limitations

  Currently, you can create a view for the remote table of a database link. When the schema of the remote table changes, the view is recompiled when being used. After recompilation upon invalidation is enabled, this process occurs regardless of the **enable_view_invalidation** setting. For example:
  a. Create **table1** in database **db1**.
  b. Create database link **dblink1** for connecting to database **db1** in database **db2**, and create view **view1** (**CREATE VIEW view1 AS SELECT * FROM table1@dblink1;**).
  c. Delete a column from table **table1** in database **db1**. When you query the view **view1** in database **db2**, the view fails to be recompiled and an error is reported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

d. Restore the deleted column of table **table1** in database **db1**. When you query the view **view1** in database **db2**, the view is successfully recompiled and the query result is returned.

NOTE

For details about recompilation after invalidation and related parameters, see **Recompilation upon Invalidation**.

- Other scenarios
  - The database link table does not support triggers in certain scenarios. For example, when a database link is used in the function called by a trigger, a trigger calls a database link function, or a trigger is defined on a database link.
  - The UPSERT and MERGE syntaxes are not supported.
  - The CURRENT CURSOR syntax is not supported.
  - Hidden columns in a table cannot be queried.
  - Database links cannot be used for remote access.
  - Database links cannot be used for direct connections to DNs.
- Limitations on dumping and backup
  - Database objects related to database links cannot be dumped. The standby node cannot be called or connected by database links.
  - Database objects such as database links cannot be restored from the backup of their clusters. The key files of different clusters are different. Therefore, you need to use the cluster key file for decryption when using database links.
- Limitations on join pushdown

  Database links do not support JOIN pushdown.
- Limitations on predicate pushdown
  - The WHERE clause can only use the built-in data types, operators, and functions of the database. The used functions must be of the IMMUTABLE type.
  - Multi-table conditions in the WHERE clause cannot be pushed down.
- Limitations on aggregate function pushdown

  Only the SELECT statement of a single table without the GROUP, ORDER BY, HAVING, or LIMIT clause is supported. Window functions are not supported.
- Hint pushdown

  Hints can be pushed down only in scan mode for database link table objects. The syntax format is as follows:

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

  The table name or table alias in a query block must be unique.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-68 Supported SQL statements

SQL Type	Operation Object	Supported Option	Execution Context
Creating a database link	Database link	-	Common transaction block, stored procedure, function, and advanced package
Modifying a database link	Database link	Only the username and password can be changed.	Common transaction block, stored procedure, function, and advanced package
Dropping a database link	Database link	-	Common transaction block, stored procedure, function, and advanced package

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQL Type	Operation Object	Supported Option	Execution Context
SELECT statement	Ordinary table, ordinary view, and complete-refresh materialized view	• WHERE clause • JOIN clause used between a database link table and an inner table • JOIN clause used between database link tables • Aggregate function • ORDER BY clause • WINDOW clause • LIMIT, OFFSET, and FETCH clauses • GROUP BY and HAVING clauses • UNION clause • WITH clause • START WITH and CONNECT BY clauses • ROWNUM usage • PIVOT clause • UNPIVOT clause • FOR UPDATE clause **NOTE** Only FOR UPDATE, FOR UPDATE NOWAIT, and FOR UPDATE WAIT are supported.	Common transaction block, stored procedure, function, advanced package, and logical view
INSERT statement	Ordinary table	• WITH clause • Multi-value insertion • RETURNING clause **NOTE** • The ON DUPLICATE KEY UPDATE clause is not supported. • The INSERT ALL statement is not supported.	Common transaction block, stored procedure, function, and advanced package
UPDATE statement	Ordinary table	• WITH clause • LIMIT clause • ORDER BY clause • WHERE clause • RETURNING clause **NOTE** The multi-table UPDATE operation is not supported.	Common transaction block, stored procedure, function, and advanced package

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQL Type	Operation Object	Supported Option	Execution Context
DELETE statement	Ordinary table	• WITH clause • LIMIT clause • ORDER BY clause • WHERE clause • RETURNING clause **NOTE** The multi-table DELETE operation is not supported.	Common transaction block, stored procedure, function, and advanced package
LOCK TABLE statement	Ordinary table	• LOCKMODE clause • NOWAIT clause	Common transaction block

NOTE

For partitioned tables, the PARTITION clause can be used to specify partitions.

Table 7-69 Supported GaussDB database table types in non-multi-primary mode

Dimension	GaussDB Table Type		Database Links
TEMP option	Temporary table		Not supported
	Global temporary table		Not supported
UNLOGGED option	Unlogged table		Supported
Storage feature	Row store	Astore	Supported
		Ustore	Not supported
	Partitioned table		Not supported
	Subpartitioned table		Not supported
View	Remote view accessed through a database link		SELECT supported; DML not
	Remote table joined with a local view through a database link		SELECT supported; DML not

Examples

Establishing a connection to GaussDB:

```
-- DDL statements related to database links
-- Perform pre-operations.
gaussdb=# CREATE USER jack WITH PASSWORD '********';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CREATE ROLE
-- Grant the permission to create database links to a user.
gaussdb=# GRANT CREATE PUBLIC DATABASE LINK TO jack;
GRANT
-- Grant the permission to drop database links to a user.
gaussdb=# GRANT DROP PUBLIC DATABASE LINK TO jack;
GRANT ROLE
-- Grant the permission to modify database links to a user.
gaussdb=# GRANT ALTER PUBLIC DATABASE LINK TO jack;
GRANT ROLE
-- Revoke the permission to create database links from a user.
gaussdb=# REVOKE CREATE PUBLIC DATABASE LINK FROM jack;
REVOKE
-- Revoke the permission to drop database links from a user.
gaussdb=# REVOKE DROP PUBLIC DATABASE LINK FROM jack;
REVOKE ROLE
-- Revoke the permission to modify database links from a user.
gaussdb=# REVOKE ALTER PUBLIC DATABASE LINK FROM jack;
REVOKE ROLE
-- Clean the environment.
gaussdb=# DROP USER jack;
DROP ROLE

-- Perform pre-operations.
-- Create an ORA-compatible database.
gaussdb=# CREATE DATABASE ora_test_db DBCOMPATIBILITY 'ORA';
CREATE DATABASE
-- Switch to another database.
gaussdb=# \c ora_test_db
-- Create a user with the SYSADMIN permission.
ora_test_db=# CREATE USER local_user WITH SYSADMIN PASSWORD '********';
CREATE ROLE
ora_test_db=# SET ROLE local_user PASSWORD '********';
SET
-- Create a database link. The host can also be an IPv6 address.
ora_test_db=> CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'remote_user' IDENTIFIED BY '********'
USING (host '***.***.***.***', port '*****', dbname 'remote_db');
CREATE DATABASE LINK
-- Modify the database link information.
ora_test_db=> ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'remote_user' IDENTIFIED BY '********';
ALTER DATABASE LINK
-- Drop a database link.
ora_test_db=> DROP PUBLIC DATABASE LINK dblink;
DROP DATABASE LINK

-- Clean the environment.
ora_test_db=> RESET ROLE;
RESET
ora_test_db=# DROP USER local_user;
DROP ROLE
ora_test_db=# \c postgres
gaussdb=# DROP DATABASE ora_test_db;
DROP DATABASE

-- Database link statements
-- Perform pre-operations.
gaussdb=# CREATE USER local_user WITH SYSADMIN PASSWORD '********';
CREATE ROLE
gaussdb=# CREATE USER remote_user WITH SYSADMIN PASSWORD '********';
CREATE ROLE
-- Create a remote database.
gaussdb=# CREATE DATABASE remote_db DBCOMPATIBILITY = 'ORA';
CREATE DATABASE
-- Create a database link test database.
gaussdb=# CREATE DATABASE local_db DBCOMPATIBILITY = 'ORA';
CREATE DATABASE
gaussdb=# \c remote_db
remote_db=# SET ROLE remote_user PASSWORD '********';
SET
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
remote_db=> CREATE SCHEMA remote_user;
CREATE SCHEMA
-- Create an ordinary table.
remote_db=> CREATE TABLE remote_tb(f1 int, f2 text, f3 text[]) WITH (STORAGE_TYPE = ASTORE);
CREATE TABLE
remote_db=> INSERT INTO remote_tb VALUES (0,'a','{"a0","b0","c0"}');
INSERT 0 1
remote_db=> INSERT INTO remote_tb VALUES (1,'bb','{"a1","b1","c1"}');
INSERT 0 1
remote_db=> INSERT INTO remote_tb VALUES (2,'cc','{"a2","b2","c2"}');
INSERT 0 1

-- Create a function.
remote_db=> CREATE OR REPLACE FUNCTION f(a in int, b in int)
RETURN int AS
    tmp int :=  a + b;
BEGIN
    RETURN tmp;
END;
/
CREATE FUNCTION

-- Create a synonym.
remote_db=> CREATE SYNONYM remote_sy FOR remote_tb;
CREATE SYNONYM

remote_db=> \c local_db
local_db=# SET ROLE local_user PASSWORD '********';
SET
local_db=> CREATE SCHEMA local_user;
CREATE SCHEMA
local_db=> CREATE TABLE local_tb(f1 int, f2 text, f3 text[]) WITH (STORAGE_TYPE = ASTORE);
CREATE TABLE
local_db=> INSERT INTO local_tb VALUES (2,'c','{"a2","b2","c2"}');
INSERT 0 1
local_db=> CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'remote_user' IDENTIFIED BY '********'
USING (HOST '***.***.***.***', PORT '*****', DBNAME 'remote_db'); -- Set the host and port based on the actual
situation.
CREATE DATABASE LINK
-- Query the remote table.
local_db=> SELECT * FROM remote_tb@dblink ORDER BY f1;
 f1 | f2 |     f3
----+----+------------
  0 | a  | {a0,b0,c0}
  1 | bb | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
(3 rows)
-- Insert data into the remote table.
local_db=> INSERT INTO remote_tb@dblink VALUES (4,'d','{"a1","b2","c3"}');
INSERT 0 1
-- Update the remote table.
local_db=> UPDATE remote_tb@dblink SET f2 = 'aa' WHERE f1 = 0;
UPDATE 1
-- Delete data from the remote table.
local_db=> DELETE remote_tb@dblink WHERE f1 = 1;
DELETE 1
-- Join the local table with the remote table.
local_db=> SELECT * FROM remote_tb@dblink JOIN local_tb ON local_tb.f1 = remote_tb.f1@dblink;
 f1 | f2 |     f3     | f1 | f2 |     f3
----+----+------------+----+----+------------
  2 | cc | {a2,b2,c2} |  2 | c  | {a2,b2,c2}
(1 row)
-- Aggregate function
local_db=> SELECT count(*) FROM remote_tb@dblink;
 count
-------
     3
(1 row)
-- Access a remote function.
```

```
local_db=> SELECT f@dblink(1,2);
 f
---
 3
(1 row)
-- Access a remote function in PLSQL_BODY.
local_db=> CREATE OR REPLACE FUNCTION call_f(a in int, b in int)
RETURN INT AS
    tmp int;
BEGIN
    tmp := f@dblink(a, b);
    RETURN tmp;
END;
/
CREATE FUNCTION
local_db=> SELECT call_f(1, 2);
 call_f
--------
      3
(1 row)
-- Create a synonym for a database link.
local_db=> CREATE SYNONYM local_sy FOR remote_user.remote_tb@dblink;
CREATE SYNONYM
local_db=> SELECT * FROM local_sy ORDER BY f1;
 f1 | f2 |     f3
----+----+------------
  0 | aa | {a0,b0,c0}
  2 | cc | {a2,b2,c2}
  4 | d  | {a1,b2,c3}
(3 rows)
-- Access the synonym of the remote database.
local_db=> SELECT * FROM remote_sy@dblink ORDER BY f1;
 f1 | f2 |     f3
----+----+------------
  0 | aa | {a0,b0,c0}
  2 | cc | {a2,b2,c2}
  4 | d  | {a1,b2,c3}
(3 rows)
-- Partial hint pushdown supported by database links
local_db=> EXPLAIN (VERBOSE, COSTS off) SELECT /*+ tablescan(remote_sy) */ * FROM remote_sy@dblink;
                                 QUERY PLAN
-----------------------------------------------------------------------------------------------
 Foreign Scan on "remote_user@dblink".remote_tb remote_sy
   Output: f1, f2, f3
   Exec Nodes: (dblink#nodegroup#0) datanode1
   Remote SQL: SELECT /*+  tablescan(remote_tb) */ f1, f2, f3 FROM remote_user.remote_tb
(4 rows)
-- View the database link system catalog gs_database_link.
local_db=> SELECT dlname,dlowner,options FROM gs_database_link;
 dlname | dlowner |                       options
--------+---------+-------------------------------------------------------
 dblink |       0 | {host=***.***.***.***,port=*****,dbname=remote_db}
(1 row)

-- View the database link system view gs_db_links.
local_db=> START TRANSACTION;
START TRANSACTION
local_db=> SELECT * FROM remote_sy@dblink ORDER BY f1;
 f1 | f2 |     f3
----+----+------------
  0 | aa | {a0,b0,c0}
  2 | cc | {a2,b2,c2}
  4 | d  | {a1,b2,c3}
(3 rows)
local_db=> SELECT intransaction FROM gs_db_links;
 intransaction
---------------
 t
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
local_db=> END;
COMMIT

-- Clean the environment.
local_db=> \c postgres
gaussdb=# DROP DATABASE local_db;
DROP DATABASE
gaussdb=# DROP DATABASE remote_db;
DROP DATABASE
gaussdb=# DROP USER local_user;
DROP ROLE
gaussdb=# DROP USER remote_user;
DROP ROLE
```

7.36.4 Typical Issues

Q1: Why does a database link keep waiting for a lock to be released when querying remote data?

Symptom: When a database link queries remote data, it keeps waiting for a lock.

Cause analysis: If a long transaction uses a database link to operate a remote object for the first time, the transaction holds the lock until it ends. Other transactions that use the database link for the first time are blocked.

Solution: Execute a fast statement to query the remote object to be used so that the metadata of the remote object is cached locally.

7.37 Single-Node Table

7.37.1 Overview

A single-node table is an optimization solution employed by distributed databases in enterprise-grade application environments for tables with limited data volumes and sharding challenges. It exhibits the following core characteristics:

- Its design adopts the distribution types of replication tables.
- Metadata definitions are distributed globally (visible to all nodes), and table definition information is synchronized globally to all nodes to ensure cross-node SQL parsing.
- Data entities are stored exclusively on a specified DN.

7.37.2 Technical Principles

Single-Node Table and Node Group

- Basic binding rules

  Single-node tables must and can only be created in system-preset single-node groups. For details, see **Table 7-70**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 7-70 Core attributes of single-node groups

Attribute Type	Description
Type identifier	Fixed at **'s'**.
Naming conventions	Must be prefixed with **sys_single_group** and suffixed with a number (for example, **sys_single_group1** and **sys_single_group2**).
Member restrictions	Contains only one DN (primary) to form a physically isolated single-node storage unit.
Creation method	The system automatically creates a single-node group for each DN. Single-node tables must be created in these system-preset single-node groups.

- Version upgrade compatibility specifications (from earlier versions to 507.0)
  - Upgrade pre-check phase:
    - Conflict detection:

      Scan all user-defined node groups and check for any node groups illegally named with the reserved prefix **sys_single_group**.
    - Countermeasure:

      If a conflict is detected (that is, a node group is illegally named with the reserved prefix **sys_single_group**), stop the upgrade process immediately and modify the conflicting node group.

      ```
      ALTER NODE GROUP sys_single_group_userdefined RENAME TO custom_group1;
      ```

      If no conflict is detected, the pre-check passes and the upgrade proceeds.
  - Upgrade implementation phase:
    - Automatically generate compliant single-node groups for all DNs (node type: **'D'**).
    - Naming convention: **sys_single_group***${seq}* (for example, DN2 corresponds to **sys_single_group2**).
    - Establish metadata mapping to cascade tables.

Core DDL Support Mechanism with Single-Node Tables

1. In single-node table configurations, there are two key **reloptions** parameters:
   - **is_single_node**: specifies whether the table is a single-node table.
   - **is_only_meta**: specifies whether the current DN stores only metadata, not actual data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

When a DML operation is executed on a single-node table hosted on a metadata-only DN:

- If the expression in the operation is neither **false** nor directly optimizable to **false** (for example, an IS NULL condition applied to a NOT NULL column), the system reports an error.
- If the expression in the operation is **false** or can be directly optimized to **false**, the system does not report an error but returns an empty result.

2. The creation policy for single-node tables is controlled by the following GUC parameters:
   - **single_node_table_strategy**: defines the behavior of creating a single-node table.
   - **default_nodegroup**: specifies whether to set the single-node attribute by default during schema creation.

   NOTE

   For specific usage of these parameters, refer to "Database GUC Parameters > GUC Parameters > Single-Node Table" in *Reference*.
3. Only online conversion between single-node tables and distributed tables is supported.

Core DML Support Mechanism with Single-Node Tables

1. Single-node tables reuse the distributed attributes of replication tables, involving adaptations primarily in the SQL parser, optimizer, and executor.
2. After the SQL engine parses the syntax and generates a syntax tree, it proceeds to semantic parsing. During this phase, it conducts semantic checks on the parse tree (to verify semantic rules such as table existence and column data type compatibility), builds a range table for the single-node table, and ultimately generates a logical query tree. The table type (single-node or distributed) and node group are identified during query tree generation.
3. The distributed plans for single-node tables support statement pushdown (LightProxy and FQS plans), plan pushdown (Stream plan), and partial statement pushdown (XC plan). To determine whether an SQL statement can be directly pushed down to DNs, the system traverses the query tree internally. For each node in the query tree, it retrieves candidate pushdown execution nodes and obtains the distribution information of the table. For a single-node table, the execution scope (DNs) is controlled via its associated single-node group. If the SQL statement involves another table (single-node or otherwise) located on a different DN—for instance, in join operations—the execution node lists of the two tables differ, preventing them from being merged onto the same DN. Consequently, the statement cannot be pushed down and will be executed using a Stream plan. In DML scenarios, the system checks the distribution information of source and target tables. If they are inconsistent, a Stream node is added to the source table.

7.37.3 Operation Guide

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.37.3.1 DDL Operations on Single-Node Tables

Specifications and Limitations

1. A single-node table can only be created as a replication table and cannot be created as a distributed table (with a distribution key specified).
2. These tables are also subject to those limitations on replication tables: GSIs cannot be created, and the table cannot be defined as a hash bucket table.
3. Single-node tables can be created only on the single-node node group preconfigured in the system.
4. Single-node tables cannot be used in M-compatible databases.
5. Single-node tables cannot be created during the observation period of an upgrade from a version that does not support single-node tables.
6. No table creation is allowed in the single-node node group preconfigured for the target version during the observation period of an upgrade from a version that does not support single-node tables.
7. If a single-node table has been created in the single-node node group preconfigured in the system, the node group cannot be scaled in.
8. Materialized views and foreign tables are not supported.
9. Logical decoding and backup & restoration can only be performed on single-node tables of homogeneous databases. If they are performed between heterogeneous databases, errors may occur.
10. Only local temporary tables, unlogged tables, global temporary tables, and partitioned tables based on replication tables are supported. Other table types are not supported.

Creating a Single-Node Table

You can create a single-node table using the methods below. Each method requires specific conditions to be met.

- Explicitly creating a single-node table
  - Set the GUC parameter **single_node_table_strategy** to a value other than **FORBID**.
  - In the CREATE TABLE statement, explicitly specify the TO GROUP group_name clause with a predefined single-node group (type ID: **'s'**) whose name is prefixed with **sys_single_group**. For example:
    ```
    CREATE TABLE table1 (id INT) TO GROUP sys_single_group1;
    ```
- Implicitly creating a single-node table
  - Omit the TO GROUP group_name clause in the CREATE TABLE statement. The system will then automatically select a node group of type **'s'** with a name prefixed by **sys_single_group**.
  - Set the GUC parameter **single_node_table_strategy**.
    - When it is set to **ALLOW_DEFAULT_NON_PART**, single-node non-partitioned tables can be implicitly created.
    - When it is set to **ALLOW_DEFAULT_ALL**, both single-node non-partitioned and single-node partitioned tables can be implicitly created.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Set the single-node attribute for the schema where the table is to be created. For details, see **Setting the Single-Node Attribute for a Schema**.

NOTE

For specific usage of the GUC parameter, refer to "Database GUC Parameters > GUC Parameters > Single-Node Table" in *Reference*.

Example syntax:

```
-- Explicitly create a single-node table.
gaussdb=# CREATE TABLE single_table(a int) DISTRIBUTE BY REPLICATION TO GROUP sys_single_group1;
CREATE TABLE

gaussdb=# \d+ single_table
                    Table "public.single_table"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: datanode1
Options: orientation=row, is_single_node=true, logical_repl_node=-1, compression=no,
storage_type=USTORE, segment=off

-- Set the GUC parameter single_node_table_strategy to allow_default_all. For details, see "Database
GUC Parameters > GUC Parameters > Single-Node Table" in Reference.
-- Verify that the value of single_node_table_strategy is allow_default_all.
gaussdb=# SHOW single_node_table_strategy;
 single_node_table_strategy
----------------------------
 allow_default_all
(1 row)

gaussdb=# CREATE SCHEMA schema1 SET NODE GROUP sys_single_group1;
CREATE SCHEMA

-- Implicitly create a single-node table.
gaussdb=# CREATE TABLE schema1.single_table1(a int);
CREATE TABLE

gaussdb=# \d+ schema1.single_table1
                       Table "schema1.single_table1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: datanode1
Options: orientation=row, is_single_node=true, logical_repl_node=-1, compression=no,
storage_type=USTORE, segment=off

gaussdb=# DROP TABLE single_table;
DROP TABLE
gaussdb=# DROP TABLE schema1.single_table1;
DROP TABLE
gaussdb=# DROP SCHEMA schema1 CASCADE;
DROP SCHEMA
```

Setting the Single-Node Attribute for a Schema

You can set the single-node attribute for a schema using the methods below. Each method requires specific conditions to be met.

- Explicitly setting the single-node attribute for a schema

In the CREATE SCHEMA statement, explicitly specify the SET NODE GROUP group_name clause with a predefined single-node group (type ID: **'s'**) whose name is prefixed with **sys_single_group**. For example:

```
CREATE SCHEMA schema1 SET NODE GROUP sys_single_group1;
```

- Implicitly setting the single-node attribute for a schema
  - Omit the SET NODE GROUP group_name clause in the CREATE SCHEMA statement. The system will then automatically select a node group of type **'s'** with a name prefixed by **sys_single_group**.
  - Create a schema with the single-node attribute under a common user. The system does not support implicitly setting the single-node attribute for schemas under the initial user or users with administrative permissions (such as OPRADMIN or CREATEROLE).
  - Set the GUC parameter **default_nodegroup** to **SCHEMA**.

NOTE

- For specific usage of the GUC parameter, refer to "Database GUC Parameters > GUC Parameters > Single-Node Table" in *Reference*.
- The CREATE SCHEMA syntax that includes the CREATE TABLE clause (referred to as a composite SQL statement) does not support the single-node table feature. This means it cannot be used to create single-node tables or set the single-node table attribute for schemas. For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE SCHEMA" in *Reference*.

Example syntax:

```
-- Create a schema and explicitly set the single-node attribute.
gaussdb=# CREATE SCHEMA schema1 SET NODE GROUP sys_single_group1;
CREATE SCHEMA

gaussdb=# SELECT * FROM pg_namespace WHERE nspname='schema1';
 nspname | nspowner | nsptimeline | nspacl | in_redistribution | nspblockchain | nspcollation | nspgroupoid
---------+----------+-------------+--------+-------------------+---------------+--------------+-------------
 schema1 |       10 |           0 |        | n                 | f             |              |       16389
(1 row)

gaussdb=# DROP SCHEMA schema1 CASCADE;
DROP SCHEMA

gaussdb=# CREATE USER test_usr PASSWORD "********";
CREATE ROLE

gaussdb=# GRANT ALL ON DATABASE POSTGRES TO test_usr;
GRANT

-- Switch to the test_usr user.
-- Set the GUC parameter default_nodegroup to schema. For details, see "Database GUC Parameters > GUC Parameters > Single-Node Table" in Reference.
-- Verify that the value of default_nodegroup is schema.
gaussdb=> SHOW default_nodegroup;
 default_nodegroup
---------------------------
 schema
(1 row)

-- Create a schema and implicitly set the single-node attribute.
gaussdb=> CREATE SCHEMA schema2;
CREATE SCHEMA

gaussdb=> SELECT * FROM pg_namespace WHERE nspname='schema2';
 nspname | nspowner | nsptimeline | nspacl | in_redistribution | nspblockchain | nspcollation | nspgroupoid
---------+----------+-------------+--------+-------------------+---------------+--------------+-------------
 schema2 |    16395 |           0 |        | n                 | f             |              |       16390
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# DROP SCHEMA schema2 CASCADE;
DROP SCHEMA
```

Modifying the Single-Node Attribute of a Schema

- To modify the single-node attribute, do as follows:

  In the ALTER SCHEMA statement, specify the SET NODE GROUP group_name clause with a predefined single-node group (type ID: **'s'**) whose name is prefixed with **sys_single_group**. For example:

  ```
  ALTER SCHEMA schema1 SET NODE GROUP sys_single_group1;
  ```

- To delete the single-node attribute, do as follows:

  Specify the DROP NODE GROUP clause in the ALTER SCHEMA statement.

Example syntax:

```
gaussdb=# CREATE SCHEMA schema1;
CREATE SCHEMA

gaussdb=# SELECT * FROM pg_namespace WHERE nspname='schema1';
 nspname | nspowner | nsptimeline | nspacl | in_redistribution | nspblockchain | nspcollation | nspgroupoid
---------+----------+-------------+--------+-------------------+---------------+--------------+-------------
 schema1 |       10 |           0 |        | n                 | f             |              |
(1 row)

-- Modify the single-node attribute of the schema.
gaussdb=# ALTER SCHEMA schema1 SET NODE GROUP sys_single_group1;
ALTER SCHEMA

gaussdb=# SELECT * FROM pg_namespace WHERE nspname='schema1';
 nspname | nspowner | nsptimeline | nspacl | in_redistribution | nspblockchain | nspcollation | nspgroupoid
---------+----------+-------------+--------+-------------------+---------------+--------------+-------------
 schema1 |       10 |           0 |        | n                 | f             |              |       16389
(1 row)

-- Delete the single-node attribute of the schema.
gaussdb=# ALTER SCHEMA schema1 DROP NODE GROUP;
ALTER SCHEMA

gaussdb=# SELECT * FROM pg_namespace WHERE nspname='schema1';
 nspname | nspowner | nsptimeline | nspacl | in_redistribution | nspblockchain | nspcollation | nspgroupoid
---------+----------+-------------+--------+-------------------+---------------+--------------+-------------
 schema1 |       10 |           0 |        | n                 | f             |              |
(1 row)

gaussdb=# DROP SCHEMA schema1 CASCADE;
DROP SCHEMA
```

7.37.3.2 Online Conversion of Single-Node Tables

Specifications and Limitations

1. A single-node table can only be created as a replication table and cannot be created as a distributed table (with a distribution key specified).
2. GSIs cannot be created on single-node tables. After a distributed table with GSIs is converted to a single-node table, the GSIs are transformed into ordinary unique indexes. Conversely, when a single-node table is converted to a hash-distributed table, unique indexes that do not include a distribution key are transformed into GSIs.
3. Only conversion between single-node tables and distributed tables is supported; modifying distribution keys is not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4. Logical replication is not supported during the conversion between single-node tables and distributed tables.
5. Only online conversion between single-node tables and distributed tables is supported; offline conversion is not available.
6. When a single-node table is converted to a distributed table, it can only be converted to a hash-, range-, or list-distributed table. Similarly, only hash-, range-, and list-distributed tables can be converted to single-node tables.
7. Single-node tables can only be created in predefined single-node groups in the system.
8. Online conversion of single-node tables is limited to ordinary tables and partitioned tables. Other table types—including subpartitioned, segment-page, hash bucket, temporary, and unlogged tables—are not supported.
9. When a single-node table is converted to a distributed table or vice versa, DDL operations may involve concurrent reads and writes. Since long-running transactions can block online DDL operations, avoid performing online DDL operations while long-running transactions are active. The later phase of this feature will block DQL and DML operations.
10. This feature is also subject to the limitations of the online DDL feature.

Prerequisites

The GUC parameter **single_node_table_strategy** is not set to **FORBID**. For specific usage of the GUC parameter, refer to "Database GUC Parameters > GUC Parameters > Single-Node Table" in *Reference*.

Examples

- Converting a distributed table to a single-node table

```
-- Create a distributed table.
gaussdb=# CREATE TABLE test (c1 int , c2 int, c3 numeric(64,8)) WITH (storage_type = astore)
DISTRIBUTE BY HASH(c2) PARTITION BY RANGE (c2)
        (PARTITION P1 VALUES LESS THAN (5),
        PARTITION P2 VALUES LESS THAN (10),
        PARTITION P3 VALUES LESS THAN (20));
CREATE TABLE
gaussdb=# \d+ test
                        Table "public.test"
 Column |     Type      | Modifiers | Storage | Stats target | Description
--------+---------------+-----------+---------+--------------+-------------
 c1     | integer       |           | plain   |              |
 c2     | integer       |           | plain   |              |
 c3     | numeric(64,8) |           | main    |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Distribute By: HASH(c2)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=astore, compression=no

-- Convert the distributed table to a single-node table.
gaussdb=# ALTER TABLE ONLINE test DISTRIBUTE BY REPLICATION TO GROUP sys_single_group1;
ALTER TABLE
-- View the single-node table.
gaussdb=# \d+ test
                        Table "public.test"
 Column |     Type      | Modifiers | Storage | Stats target | Description
--------+---------------+-----------+---------+--------------+-------------
 c1     | integer       |           | plain   |              |
 c2     | integer       |           | plain   |              |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 c3     | numeric(64,8) |           | main    |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: datanode1
Options: orientation=row, storage_type=astore, compression=no, is_single_node=true,
logical_repl_node=-1

gaussdb=# DROP TABLE IF EXISTS test;
DROP TABLE
```

- Converting a single-node table to a distributed table

```
-- Create a single-node table.
gaussdb=#  DROP TABLE IF EXISTS test;
DROP TABLE
gaussdb=# CREATE TABLE test (c1 int , c2 int, c3 numeric(64,8)) WITH (storage_type = astore)
DISTRIBUTE BY REPLICATION TO GROUP sys_single_group2 PARTITION BY RANGE (c2)
        (PARTITION P1 VALUES LESS THAN (5),
        PARTITION P2 VALUES LESS THAN (10),
        PARTITION P3 VALUES LESS THAN (20));
CREATE TABLE
gaussdb=# \d+ test
                        Table "public.test"
 Column |     Type      | Modifiers | Storage | Stats target | Description
--------+---------------+-----------+---------+--------------+-------------
 c1     | integer       |           | plain   |              |
 c2     | integer       |           | plain   |              |
 c3     | numeric(64,8) |           | main    |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: datanode2
Options: orientation=row, storage_type=astore, is_single_node=true, logical_repl_node=-2,
compression=no

-- Convert the single-node table to a distributed table.
gaussdb=# ALTER TABLE ONLINE test DISTRIBUTE BY HASH(c2);
ALTER TABLE
-- View the distributed table.
gaussdb=# \d+ test
                        Table "public.test"
 Column |     Type      | Modifiers | Storage | Stats target | Description
--------+---------------+-----------+---------+--------------+-------------
 c1     | integer       |           | plain   |              |
 c2     | integer       |           | plain   |              |
 c3     | numeric(64,8) |           | main    |              |
Partition By RANGE(c2)
Number of partitions: 3 (View pg_partition to check each partition range.)
Has OIDs: no
Distribute By: HASH(c2)
Location Nodes: ALL DATANODES
Options: orientation=row, storage_type=astore, compression=no

gaussdb=#  DROP TABLE IF EXISTS test;
DROP TABLE
```

7.37.3.3 DML Operations on Single-Node Tables

Prerequisites

- To determine whether the current table is a correct single-node table, perform the following steps:

```
-- Create a single-node table named t_ng1.
gaussdb=# CREATE TABLE t_ng1(c1 int, c2 int) DISTRIBUTE BY REPLICATION TO GROUP
sys_single_group1;
gaussdb=# \d+ t_ng1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                    Table "public.t_ng1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | integer |           | plain   |              |
 c2     | integer |           | plain   |              |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: datanode1
Options: orientation=row, is_single_node=true, logical_repl_node=-1, compression=no,
storage_type=USTORE, segment=off

gaussdb=# SELECT reloptions FROM pg_class WHERE relname = 't_ng1';
                                            reloptions
---------------------------------------------------------------------------------------------------------------

{orientation=row,is_single_node=true,logical_repl_node=-1,compression=no,storage_type=USTORE,seg
ment=off}
(1 row)
```

- To determine whether the current single-node table was created in **sys_single_group** (single-node group), perform the following steps:

```
-- Query the pgxc_class system catalog to identify the node group of the single-node table. group
name must be sys_single_group[%d].
gaussdb=# SELECT pgroup, nodeoids FROM pgxc_class WHERE pcrelid = (SELECT oid FROM pg_class
WHERE relname = 't_ng1');
      pgroup       | nodeoids
-------------------+----------
 sys_single_group1 | 16385
(1 row)
gaussdb=# DROP TABLE t_ng1;
```

Specifications and Limitations

- Specifications:
  a. Single-node tables support standard DML statements: INSERT, UPDATE, DELETE, UPSERT, MERGE, SELECT, and SELECT FOR UPDATE | SHARED. (FOR UPDATE | SHARED is subject to the same limitations as Ustore/Astore.)
  b. Single-node tables can interact with non-single-node tables (such as distributed or partitioned tables) via DML. Here, distributed tables include replication tables, range-distributed tables, list-distributed tables, hash-distributed tables, hash bucket tables, and single-node tables; partitioned tables include both partitioned and subpartitioned tables.
  c. Single-node tables support SMP and TRIGGER, and can be used within stored procedures.
  d. The distribution attributes and functional specifications of single-node tables are identical to those of replication tables.
  e. After a remote single-node table is accessed via a database link (for example, through DML operations), the **ftoptions** column in the pg_foreign_table system catalog will be flagged with **is_single_datanode_table=true**. (This flag is exclusive to single-node tables.) If the table is later converted to another type, the **ftoptions** column in pg_foreign_table is updated only upon subsequent DML operations.
- Limitations:
  a. In the MERGE INTO USING clause, if both the target and source tables are single-node tables, they must belong to the same single-node group.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

b. INSERT ALL can be executed on single-node tables, but all of its target tables must be single-node tables from the same single-node group.

c. Database links can be used to query or perform DML operations on remote single-node tables. However, once an SQL statement establishes a database link to a single-node table, it cannot simultaneously include a database link to a non-single-node table.

d. When a distributed table and a single-node table are joined in the order specified by the /*+ stream */ hint, the optimizer may disregard the hint. This occurs because the optimizer automatically evaluates execution costs—if a lower-cost plan is identified, it will be prioritized over the hinted plan.

e. When statistics on single-node partitioned tables are collected in cascading mode with **PARTITION_MODE** set to **ALL**, the behavior is converted to the **ALL COMPLETE** mode.

f. Single-node tables are available only within single-node groups.

g. SMP operations are not supported when single-node tables interact with non-single-node tables.

h. SMP operations are not supported when a single-node table interacts with another single-node table located on a different DN.

Examples

```
-- sys_single_group1 and sys_single_group2 are single-node groups.
-- Create single-node tables.
gaussdb=# CREATE TABLE t_sig1(c1 int, c2 int, c3 int) TO GROUP sys_single_group1;
gaussdb=# CREATE TABLE t_sig2(c1 int primary key, c2 int, c3 int) TO GROUP sys_single_group2;
-- Insert data into a single-node table.
gaussdb=# INSERT INTO t_sig1 VALUES(generate_series(1, 20), generate_series(21, 40), generate_series(1, 20));
```

Scenario: Typical DML operations on a single-node table

```
-- SELECT on a single-node table.
gaussdb=# SELECT t_sig1.* FROM t_sig1 WHERE t_sig1.c1 > 5 AND (t_sig1.*)::t_sig1 IS NOT NULL ORDER BY 1 LIMIT 5;
 c1 | c2 | c3
----+----+----
  6 | 26 |  6
  7 | 27 |  7
  8 | 28 |  8
  9 | 29 |  9
 10 | 30 | 10
(5 rows)
-- SELECT FOR UPDATE | SHAR on a single-node table.
gaussdb=# SELECT t_sig1.* FROM t_sig1 WHERE t_sig1.c1 > 15 ORDER BY 1, 2 FOR UPDATE;
 c1 | c2 | c3
----+----+----
 16 | 36 | 16
 17 | 37 | 17
 18 | 38 | 18
 19 | 39 | 19
 20 | 40 | 20
(5 rows)
gaussdb=# SELECT t_sig1.* FROM t_sig1 WHERE t_sig1.c1 > 15 ORDER BY 1, 2 FOR SHARE;
 c1 | c2 | c3
----+----+----
 16 | 36 | 16
 17 | 37 | 17
 18 | 38 | 18
 19 | 39 | 19
 20 | 40 | 20
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(5 rows)
-- INSERT SELECT on a single-node table.
gaussdb=# INSERT INTO t_sig2 SELECT * FROM t_sig1;
-- UPSERT on a single-node table.
gaussdb=# INSERT INTO t_sig1 VALUES (1, 21) ON duplicate key UPDATE c2 = c2 + 100;
-- UPDATE on a single-node table.
gaussdb=# UPDATE t_sig1 SET c2 = (SELECT t_sig1.c1 FROM t_sig1 WHERE t_sig1.c2 = 20) WHERE c1 = 100;
-- DELETE on a single-node table.
gaussdb=# DELETE FROM t_sig1 WHERE t_sig1.c1 = 100;
-- MERGE on a single-node table.
gaussdb=# CREATE TABLE t_sig11(c1 int, c2 int, c3 int) TO GROUP sys_single_group1;
gaussdb=# INSERT INTO t_sig11 VALUES(100, 100, 100), (101, 101, 101);
gaussdb=# MERGE INTO t_sig1 AS mt1 USING t_sig11 AS mt2 ON mt1.c1 = mt2.c1 WHEN MATCHED THEN
UPDATE SET mt1.c2 = 100 WHEN NOT MATCHED THEN INSERT (c1, c2, c3) VALUES (100, 200, 300);
-- Drop single-node tables.
gaussdb=# DROP TABLE t_sig1;
gaussdb=# DROP TABLE t_sig2;
gaussdb=# DROP TABLE t_sig11;
```

Scenario: Interaction between single-node tables and distributed tables

```
-- Create tables.
gaussdb=# CREATE TABLE t_rep(c1 int, c2 int) DISTRIBUTE BY replication; -- Replication table
gaussdb=# CREATE TABLE t_dis(c1 int, c2 int) DISTRIBUTE BY hash(c1); -- Hash-distributed table
gaussdb=# CREATE TABLE t_sig1(c1 int, c2 int) DISTRIBUTE BY replication TO GROUP sys_single_group1; --
Single-node table
gaussdb=# CREATE TABLE t_sig2(c1 int, c2 int) DISTRIBUTE BY replication TO GROUP sys_single_group2; --
Single-node table
gaussdb=# CREATE TABLE t_sig11(c1 int, c2 int) DISTRIBUTE BY replication TO GROUP sys_single_group1;--
Single-node table
gaussdb=# INSERT INTO t_rep SELECT v,v FROM generate_series(1,10) AS v;
gaussdb=# INSERT INTO t_sig1 VALUES (generate_series(11, 20), generate_series(11, 20));
gaussdb=# INSERT INTO t_dis VALUES (generate_series(11, 20), generate_series(11, 20));
gaussdb=# INSERT INTO t_sig2 VALUES (generate_series(11, 20), generate_series(11, 20));
-- UNION all.
gaussdb=# SELECT * FROM t_sig1 UNION all SELECT * FROM t_sig2 ORDER BY 1;
 c1 | c2
----+----
 11 | 11
 11 | 11
 12 | 12
 12 | 12
 13 | 13
 13 | 13
 14 | 14
 14 | 14
 15 | 15
 15 | 15
 16 | 16
 16 | 16
 17 | 17
 17 | 17
 18 | 18
 18 | 18
 19 | 19
 19 | 19
 20 | 20
 20 | 20
(20 rows)
gaussdb=# SELECT * FROM t_sig1 UNION all SELECT * FROM t_rep ORDER BY 1 LIMIT 5;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  5 |  5
(5 rows)
gaussdb=# SELECT * FROM t_sig1 UNION ALL SELECT * FROM t_dis ORDER BY 1 LIMIT 5 OFFSET 2;
 c1 | c2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
----+----
 12 | 12
 12 | 12
 13 | 13
 13 | 13
 14 | 14
(5 rows)
-- Join.
gaussdb=# SELECT * FROM t_sig1 LEFT JOIN t_dis ON t_sig1.c1 = t_dis.c1 ORDER BY 1,2 LIMIT 5;
 c1 | c2 | c1 | c2
----+----+----+----
 11 | 11 | 11 | 11
 12 | 12 | 12 | 12
 13 | 13 | 13 | 13
 14 | 14 | 14 | 14
 15 | 15 | 15 | 15
(5 rows)
gaussdb=# SELECT * FROM t_sig1 LEFT JOIN t_rep ON t_sig1.c1 = t_rep.c1 + 10 ORDER BY 1,2 LIMIT 5;
 c1 | c2 | c1 | c2
----+----+----+----
 11 | 11 |  1 |  1
 12 | 12 |  2 |  2
 13 | 13 |  3 |  3
 14 | 14 |  4 |  4
 15 | 15 |  5 |  5
(5 rows)
-- Pushdown scenarios for single-node tables within the same node group:
-- Push down SELECT.
gaussdb=# EXPLAIN (costs off) SELECT * FROM t_sig1 WHERE c1 IN (SELECT c1 FROM t_sig11 WHERE c2 >
10 LIMIT 2);
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Node/s: datanode1
(2 rows)
-- Push down INSERT SELECT.
gaussdb=# EXPLAIN (costs off) INSERT INTO t_sig1 SELECT * FROM t_sig11 WHERE c1 > 10;
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Node/s: (sys_single_group1) datanode1
(2 rows)
-- Push down UPDATE.
gaussdb=# EXPLAIN (costs off) UPDATE t_sig1 SET (c1, c2) = (SELECT c1 + 10, c2 - c1 FROM t_sig11);
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Node/s: (sys_single_group1) datanode1
(2 rows)
-- Push down DELETE.
gaussdb=# EXPLAIN (costs off) DELETE FROM t_sig1 WHERE c1 = (SELECT c1 FROM t_sig11 WHERE c2 >
10);
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Node/s: (sys_single_group1) datanode1
(2 rows)
-- Push down JOIN.
gaussdb=# EXPLAIN (costs off) SELECT * FROM t_sig1, t_sig11 WHERE t_sig1.c1 = t_sig11.c2;
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Node/s: (sys_single_group1) datanode1
(2 rows)
-- Push down UNION ALL.
gaussdb=# EXPLAIN (costs off) SELECT * FROM t_sig1 UNION ALL SELECT * FROM t_sig11 LIMIT 4;
                QUERY PLAN
-------------------------------------------
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  Node/s: (GenGroup) datanode1
(2 rows)

-- SMP scenarios for single-node tables:
-- Note: In the debug version, set query_dop to 1002 to forcibly enable SMP.
gaussdb=# SET query_dop = 1002;
gaussdb=# SET enable_fast_query_shipping = off;
-- Range query.
gaussdb=# EXPLAIN (costs off) SELECT * FROM t_sig1;
                QUERY PLAN
-----------------------------------------------
 Streaming (type: GATHER)
   Node/s: (GenGroup) datanode1
   ->  Streaming(type: LOCAL GATHER dop: 1/2)
         Spawn ON: (sys_single_group1) datanode1
         ->  Seq Scan ON t_sig1
(5 rows)
-- Aggregate.
gaussdb=# EXPLAIN (costs off) SELECT count(*) FROM t_sig1;
                  QUERY PLAN
-----------------------------------------------------
 Streaming (type: GATHER)
   Node/s: (GenGroup) datanode1
   ->  Aggregate
         ->  Streaming(type: LOCAL GATHER dop: 1/2)
               Spawn ON: (sys_single_group1) datanode1
               ->  Aggregate
                     ->  Seq Scan ON t_sig1
(7 rows)
-- Point query.
gaussdb=# EXPLAIN (costs off) SELECT avg(c1) as avg, c1 FROM t_sig1 WHERE c1 > 15 GROUP BY c1
HAVING c1 < 19 ORDER BY 1, 2;
                        QUERY PLAN
-----------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: (GenGroup) datanode1
   ->  Sort
         Sort Key: (pg_catalog.avg((avg(c1)))), c1
         ->  Streaming(type: LOCAL GATHER dop: 1/2)
               Spawn ON: (sys_single_group1) datanode1
               ->  HashAggregate
                     GROUP BY Key: c1
                     ->  Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)
                           Spawn ON: (sys_single_group1) datanode1
                           ->  HashAggregate
                                 GROUP BY Key: c1
                                 ->  Seq Scan ON t_sig1
                                       Filter: ((c1 > 5) AND (c1 < 9))
(14 rows)
-- Drop tables.
gaussdb=# DROP TABLE t_rep;
gaussdb=# DROP TABLE t_dis;
gaussdb=# DROP TABLE t_sig1;
gaussdb=# DROP TABLE t_sig2;
gaussdb=# DROP TABLE t_sig11;
```

7.37.4 FAQ

Symptom: An error occurs when users execute a DML statement involving read/write operations on a single-node table.

Cause analysis: The data of a single-node table is stored only on the DN associated with the designated **sys_single_group**. If the scan or modifytable operator for this table is pushed down to a DN that does not actually store the table's data, an error is triggered to block the operation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Solution: Use EXPLAIN to verify that the execution plan is appropriate. Alternatively, verify that the operator is pushed down to the correct DN (the one that stores the data of the single-node table).

7.38 ROWID Table

7.38.1 Overview

A ROWID table is defined as a table containing the system column **ROWID** or **ROWNO**. **ROWID** is not a conventional database column but a pseudocolumn that uniquely identifies each row in the table. **ROWID** provides direct access to the physical locations of rows and is automatically owned by each row without requiring explicit definition in the table. Different tables may contain identical **ROWID** values, which cannot be modified through INSERT, UPDATE, or DELETE operations. The ROWID table feature supports the following functionalities:

- Creating and dropping ROWID tables, including CREATE TABLE AS and CREATE TABLE LIKE.
- Maintaining the system column **ROWID** or **ROWNO** during DML operations on ROWID tables.
- Maintaining ROWID tables, including VACUUM FULL, CLUSTER, and ALTER TABLE SET TABLESPACE.
- Converting a non-ROWID table to a ROWID table using ALTER TABLE SET WITH ROWID.
- Supporting features that use ROWID tables, such as cross-page transparent compression and flashback.

NOTE

- The **ROWID** column records the table's node and OID information. Its value may change upon alternations to the table's distribution or partitioning configuration.
  - Alternations to distribution configuration: for example, updating the distribution key, redistributing the table during scale-out or scale-in, or converting between single-node and distributed table types.
  - Alternations to partitioning configuration: for example, updating the partition key (which leads to cross-partitioning), or performing DDL operations (EXCHANGE, MERGE, SPLIT, or MOVE) on the partitioned table.
- Trigger-type variables are unable to access the system column **ROWID** or **ROWNO**.
- If a range table (subquery)—for example, a Dual table—does not include the **ROWID** column, any query referencing it will generate an error.
- The system column **ROWID** is unavailable when DNs are directly connected.
- When tuples are inserted into a ROWID table (including indirect insertion scenarios such as COPY INTO and VACUUM FULL), the sequence of the system column **ROWNO** is read. In distributed mode, however, this sequence must be obtained from the GTM, which results in performance degradation. The performance impact becomes even more pronounced in high-concurrency scenarios.
- When fine-grained backup and restoration are performed on a ROWID table, the value in the **ROWID** column of the new table may differ from that in the original table.

7.38.2 Operation Guide

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.38.2.1 DDL

Create and drop a ROWID table (using a non-segment-page, non-partitioned Astore table as an example):

```
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".
-- Step 1: Create a ROWID table.
ora=# CREATE TABLE t1 (c1 INT) WITH (storage_type = ASTORE, segment = OFF, hasrowid = ON);
CREATE TABLE
-- Step 2: Observe objects. A ROWNO sequence and ROWNO index are created.
ora=# \d
                                        List of relations
 Schema |  Name  |   Type   | Owner |                                  Storage
--------+--------+----------+-------+-------------------------------------------------------------------------------
 public | t1     | table    | Mike  |
{orientation=row,storage_type=astore,segment=off,hasrowid=on,compression=no}
 public | t1_seq | sequence | Mike  |
(2 rows)
ora=# \di
                   List of relations
 Schema |     Name      | Type  | Owner | Table | Storage
--------+---------------+-------+-------+-------+---------
 public | t1_rowno_idx | index | Mike  | t1    |
(1 row)
-- Step 3: Insert data and query ROWID.
ora=# INSERT INTO t1 VALUES (1), (2), (3);
INSERT 0 3
ora=# SELECT rowid, rowno, * FROM t1;
             rowid              | rowno | c1
--------------------------------+-------+----
 AAAAAAAAAP//AAAHvoAAAAAAAAAAB |     1 |  1
 AAAAAAAAAP//AAAHvoAAAAAAAAAAC |     2 |  2
 AAAAAAAAAP//AAAHvoAAAAAAAAAAD |     3 |  3
(3 rows)
ora=# SELECT rowid, rowno, * FROM t1 WHERE rowid = 'AAAAAAAAAP//AAAHvoAAAAAAAAAAC';
             rowid              | rowno | c1
--------------------------------+-------+----
 AAAAAAAAAP//AAAHvoAAAAAAAAAAC |     2 |  2
(1 row)
-- Step 4: Drop the ROWID table.
ora=# DROP TABLE t1;
DROP TABLE
-- Step 5: Observe objects. The ROWNO sequence and ROWNO index have been automatically removed.
ora=# \d
No relations found.
ora=# \di
No relations found.
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

Convert a non-ROWID table to a ROWID table (using a non-segment-page, non-partitioned Astore table as an example):

```
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
-- Step 1: Create a non-ROWID table.
ora=# CREATE TABLE t1 (c1 INT) WITH (storage_type = ASTORE, segment = OFF);
CREATE TABLE
-- Step 2: Observe objects. Only the table exists at this stage.
ora=# \d
                                List of relations
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Schema | Name | Type  | Owner |                          Storage
--------+------+-------+-------+-------------------------------------------------------------------------
 public | t1   | table | Mike  | {orientation=row,storage_type=astore,segment=off,compression=no}
(1 row)

ora=# \di
No relations found.
-- Step 3: Convert the non-ROWID table to a ROWID table.
ora=# ALTER TABLE t1 SET WITH ROWID;
ALTER TABLE
-- Step 4: Observe objects. A ROWNO sequence and ROWNO index are created.
ora=# \d
                                    List of relations
 Schema |  Name  |   Type   | Owner |                                Storage
--------+--------+----------+-------+-----------------------------------------------------------------------------------------------
 public | t1     | table    | Mike  |
{orientation=row,storage_type=astore,segment=off,compression=no,hasrowid=on}
 public | t1_seq | sequence | Mike  |
(2 rows)

ora=# \di
                  List of relations
 Schema |     Name      | Type  | Owner | Table | Storage
--------+---------------+-------+-------+-------+---------
 public | t1_rowno_idx | index | Mike  | t1    |
(1 row)
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

For details about the DDL syntax, see "SQL Reference > SQL Syntax" in *Reference*.

7.38.2.2 ROWID Type

A ROWID contains the essential information required to locate a row within a table, enabling rapid access to specific data records.

The ROWID type is represented as a BASE64-encoded string and comprises the following details:

- Shard ID, indicating the location of the data storage node.
- Bucket ID of a hash bucket table.
- Table OID.
- Row number, which is unique for each row.

For details about the ROWID type, see "SQL Reference > Data Types > ROWID Type" in *Reference*.

7.38.2.3 DML

The system column **ROWID** can be used in DML statements, including INSERT, SELECT, UPDATE, and DELETE statements. For example:

```
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
ora=# CREATE TABLE test(a INT, b INT) WITH (hasrowid);
CREATE TABLE
ora=# -- INSERT
ora=# INSERT INTO test values(1, 1);
INSERT 0 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# -- The row inserted into the ROWID table contains the system column ROWID.
ora=# SELECT ROWID FROM test;
            rowid
------------------------------
 AAAAAAAAAP//AAAEGAAAAAAAAAAB
(1 row)
ora=# -- SELECT, using the system column ROWID as the WHERE condition.
ora=# SELECT * FROM test WHERE ROWID = 'AAAAAAAAAP//AAAEGAAAAAAAAAAB';
 a | b
---+---
 1 | 1
(1 row)
ora=# -- UPDATE, using the system column ROWID as the WHERE condition.
ora=# UPDATE test SET b = 2 WHERE ROWID = 'AAAAAAAAAP//AAAEGAAAAAAAAAAB';
UPDATE 1
ora=# -- DELETE, using the system column ROWID as the WHERE condition.
ora=# DELETE FROM test WHERE ROWID = 'AAAAAAAAAP//AAAEGAAAAAAAAAAB';
DELETE 1
ora=# -- CLEAN
ora=# DROP TABLE test;
DROP TABLE
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

When the system column **ROWID** is used in a DML statement and the **ROWID_OPT** option is enabled in the GUC parameter **opt_behavior_knob**, the system supports these query optimization scenarios: **Predicate Condition Rewriting for the System Column ROWID**, **ORDER BY Statement Rewriting for the System Column ROWID**, **Max/Min Optimization for the System Column ROWID**, and **Distribution/Partition Pruning for the System Column ROWID**. (For details about the **ROWID_OPT** option, see "Database GUC Parameters > GUC Parameters > Query Planning > Other Optimizer Options" in Reference.)

Predicate Condition Rewriting for the System Column ROWID

When the system column **ROWID** is used in the predicate condition of the WHERE or JOIN ON clause, the conditional statement can be rewritten. Currently, rewriting is limited to statements that use a comparison operator (=, <>, <, >, <=, or >=) as the predicate condition.

- If the operator is an equal sign (=), rewriting is supported in all scenarios, such as in the predicate condition "ROWID = src_expr". (Rewriting does not occur if **src_expr** is also the system column **ROWID**, as it would be meaningless). The rewriting rules are as follows:

  ```
  ROWID = src_expr AND (-1)::smallint = ROWID_HASHBUCKETID(src_expr) AND tableoid = ROWID_TABLEOID(src_expr) AND rowno = ROWID_SEQUENCE(src_expr)
  ```

  After rewriting, the first three conditions trigger distribution pruning, bucket pruning, and partition pruning, and the last condition triggers index scans to enhance scan performance. For non-rewritable expressions, for example, an IN expression like "rowid IN ($1,$2,...)", a full table scan occurs where each row of tuple is traversed for condition checks. In such cases, pruning and index scans are not applied.
- If the operator is not an equal sign (<>, <, >, <=, or >=), rewriting is supported only in some scenarios, such as the range expression "ROWID > src_expr". Since **ROWID** comparison involves internal fields (**node_slice**, **bucket_id**, **table_oid**, and **row_no**), rewriting is limited to ordinary single-node tables and requires the execution plan to be Light Proxy, FQS, or PGXC. In addition,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

src_expr must be a calculable constant. The system checks **node_id**, **table_id**, and **bucket_id** sequentially. If any field differs from the given **rowid** constant, the system directly compares it using the operator and returns a constant expression. If all fields match the given **rowid** constant, the expression is rewritten into a predicate condition for the system column **rowno**, that is:

```
rowno >/</<>/>=/<= ROWID_SEQUENCE(src_expr)
```

In non-rewritable scenarios, a full table scan is still performed.

NOTE

Predicate condition rewriting for the system column **ROWID** is subject to the following specifications and limitations:

- In the WHERE or JOIN ON condition, if the system column **ROWID** appears on both sides of the operator, rewriting does not occur.
- In constructs like ROWID IN/NOT IN (xxx) where the right-hand expression is a subquery expression, user-defined type expression, or array expression, rewriting is not supported.
- In constructs like NULLIF(ROWID,xxx) or NULLIF(ROWID,xxx), the NULLIF expression is not rewritten. However, ROWID = NULLIF(xxx,xxx) can be rewritten as usual.
- The DISTINCT FROM expression is not rewritten.
- The HAVING clause is not rewritten.
- If both sides of the operator are RECORD-type expressions, rewriting is not supported.
- The ALL/ANY/SOME expressions do not support rewriting. Judgment conditions like ROWID = ALL/ANY/SOME (xxx) or ALL/ANY/SOME (xxx) = ROWID are not rewritten.

The aforementioned limitations apply to both CNs and DNs. Whether predicate rewriting is performed depends on the actual form of the SQL statement processed by each node. If the original SQL statement does not meet the rewriting conditions on a CN, no rewriting occurs on the CN. However, if the SQL statement pushed down to a DN meets the rewriting conditions, predicate rewriting can still occur on the DN.

Example:

The value of the **tableoid** field may vary across scenarios. The following example is provided for reference only. Please refer to the actual output in case of any differences.

Assuming that the GUC parameter **explain_perf_mode** is set to **normal**:

```
-- Prepare data.
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".

ora=# CREATE TABLE t1(a int, c int) WITH (storage_type=astore, hasrowid=ON);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' AS the distribution column BY default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t1 VALUES( 1, 1);
INSERT 0 1
ora=# INSERT INTO t1 VALUES( 2,  2);
INSERT 0 1
ora=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1
ora=# INSERT INTO t1 VALUES(3, 4);
INSERT 0 1
ora=# INSERT INTO t1 VALUES(1,  5);
INSERT 0 1
ora=# INSERT INTO t1 VALUES( 2,  6);
INSERT 0 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# INSERT INTO t1 VALUES(3,  7);
INSERT 0 1
ora=# INSERT INTO t1 VALUES( 3,  8);
INSERT 0 1
ora=# INSERT INTO t1 VALUES( 1, 9);
INSERT 0 1
ora=# INSERT INTO t1 VALUES( 2, 10);
INSERT 0 1
ora=# CREATE OR REPLACE FUNCTION func RETURN rowid AS
ora$#     temp rowid;
ora$#     res int;
ora$# BEGIN
ora$#     SELECT rowid INTO temp FROM t1 ORDER BY rowid LIMIT 1;
ora$#     RETURN temp;
ora$# END;
ora$# /
CREATE FUNCTION
ora=# EXPLAIN (costs off, verbose) SELECT * FROM t1 WHERE rowid = func();
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.a, t1.c
   Node/s: All datanodes
   Remote query: SELECT a, c, rowid, tableoid, rowno FROM ONLY public.t1 WHERE true
   Coordinator quals: ((t1.rowid = func()) AND ((-1)::smallint = rowid_hashbucketid(func())) AND
(t1.tableoid = rowid_tableoid(func())) AND (t1.rowno = rowid_sequence(func())))
(5 rows)
ora=# SELECT * FROM t1 WHERE rowid = func();
 a | c
---+---
 1 | 1
(1 row)
ora=# -- The HAVING clause cannot be rewritten.
ora=# EXPLAIN (costs off, verbose) SELECT max(a),max(rowid) FROM t1 GROUP BY rowid HAVING rowid =
'AAAAAAAAAP//BBBBBAAAAAAAAAAAAB';
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                              QUERY PLAN
---------------------------------------------------------------------------------
 HashAggregate
   Output: max((max(a))), max((max(rowid))), rowid
   Group By Key: t1.rowid
   ->  Streaming (TYPE: GATHER)
         Output: (max(a)), (max(rowid)), rowid
         Node/s: datanode1
         ->  HashAggregate
               Output: max(a), max(rowid), rowid
               Group By Key: t1.rowid
               ->  Seq Scan ON public.t1
                     Output: rowid, a
                     Distribute Key: a
                     Filter: (t1.rowid = 'AAAAAAAAAP//BBBBBAAAAAAAAAAAAB'::rowid)
(13 rows)
ora=# SELECT max(a),max(rowid) FROM t1 GROUP BY rowid HAVING rowid = 'AAAAAAAAAP//
BBBBBAAAAAAAAAAAAB';
 max | max
-----+-----
(0 rows)
ora=# SELECT max(a) AS newcol1,rowid_sequence(max(rowid)) AS newcol2 FROM t1 GROUP BY rowid
ORDER BY newcol1,newcol2;
 newcol1 | newcol2
---------+---------
       1 |       1
       1 |       3
       1 |       5
       2 |       2
```

```
      2 |      4
      2 |      6
      3 |     11
      3 |     12
      3 |     13
      3 |     14
(10 rows)
ora=# -- The NULLIF expression itself is not rewritten, but rowid = NULLIF(xxx) is rewritten.
ora=# EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, rowid);
SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, rowid);
EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, 'AAAAAAAAAP//
AAAAAAAAAAAAAAAAA');
SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, 'AAAAAAAAAP//
AAAAAAAAAAAAAAAAA');
QUERY PLAN




------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------
 Data Node Scan ON "__REMOTE_FQS_QUERY__"
   Output: t1.rowid
   Node/s: All datanodes
   Remote query: SELECT rowid FROM public.t1 WHERE rowid = NULLIF(rowid, rowid) AND (-1)::smallint =
rowid_hashbucketid(NULLIF(rowid, rowid)) AND tableoid = rowid_tableoid(NULLIF(rowid, rowid)) AND
rowno
 = rowid_sequence(NULLIF(rowid, rowid))
(4 rows)
ora=# SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, rowid);
 rowid
-------
(0 rows)
ora=# EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, 'AAAAAAAAAP//
AAAAAAAAAAAAAAAAA');
                                            QUERY PLAN




------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 Data Node Scan ON "__REMOTE_FQS_QUERY__"
   Output: t1.rowid
   Node/s: All datanodes
   Remote query: SELECT rowid FROM public.t1 WHERE rowid = NULLIF(rowid, 'AAAAAAAAAP//
AAAAAAAAAAAAAAAAA'::rowid) AND (-1)::smallint = rowid_hashbucketid(NULLIF(rowid, 'AAAAAAAAAP//
AAAAAAAAAAAAAAAAA'::row
id)) AND tableoid = rowid_tableoid(NULLIF(rowid, 'AAAAAAAAAP//AAAAAAAAAAAAAAAAA'::rowid)) AND
rowno = rowid_sequence(NULLIF(rowid, 'AAAAAAAAAP//AAAAAAAAAAAAAAAAA'::rowid))
(4 rows)
ora=# SELECT  rowid FROM t1 WHERE rowid = NULLIF(rowid, 'AAAAAAAAAP//AAAAAAAAAAAAAAAAA');
            rowid
------------------------------
 AAAAAAAAAP//AAAECpAAAAAAAAAAB
 AAAAAAAAAP//AAAECpAAAAAAAAAAC
 AAAAAAAAAP//AAAECpAAAAAAAAAAD
 AAAAAAAAAP//AAAECpAAAAAAAAAAE
 AAAAAAAAAP//AAAECpAAAAAAAAAAF
 AAAAAAAAAP//AAAECpAAAAAAAAAAG
 AAAAAAAABP//AAAECpAAAAAAAAAAL
 AAAAAAAABP//AAAECpAAAAAAAAAAM
 AAAAAAAABP//AAAECpAAAAAAAAAAN
 AAAAAAAABP//AAAECpAAAAAAAAAAO
(10 rows)
ora=# -- rowexpr is not rewritten.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# EXPLAIN (costs off, verbose) SELECT a FROM t1 WHERE (rowid,c) IN (SELECT rowid,c FROM t1);
WARNING:  Statistics IN some tables OR columns(public.t1.a, public.t1.c) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                    QUERY PLAN
------------------------------------------------------------------------------------------
 Streaming (TYPE: GATHER)
   Output: public.t1.a
   Node/s: All datanodes
   ->  Hash Right Semi Join
         Output: public.t1.a
         Hash Cond: ((public.t1.rowid = public.t1.rowid) AND (public.t1.c = public.t1.c))
         ->  Streaming(TYPE: BROADCAST)
               Output: public.t1.rowid, public.t1.c
               Spawn ON: All datanodes
               Consumer Nodes: All datanodes
               ->  Seq Scan ON public.t1
                     Output: public.t1.rowid, public.t1.c
                     Distribute Key: public.t1.a
         ->  Hash
               Output: public.t1.a, public.t1.rowid, public.t1.c
               ->  Seq Scan ON public.t1
                     Output: public.t1.a, public.t1.rowid, public.t1.c
                     Distribute Key: public.t1.a
(18 rows)
ora=# SELECT a FROM t1 WHERE (rowid,c) IN (SELECT rowid,c FROM t1) ORDER BY a;
 a
---
 1
 1
 1
 2
 2
 2
 3
 3
 3
 3
(10 rows)
ora=# CREATE TYPE t1_type AS (rowid rowid, c int);
CREATE TYPE
ora=# EXPLAIN (costs off, verbose) SELECT a FROM t1 WHERE (rowid, c) IN t1_type(func(), 1);
WARNING:  Statistics IN some tables OR columns(public.t1.a, public.t1.c) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                         QUERY PLAN
-----------------------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.a
   Node/s: All datanodes
   Remote query: SELECT a, rowid FROM ONLY public.t1 WHERE c = 1
   Coordinator quals: (t1.rowid = func())
(5 rows)
ora=# SELECT a FROM t1 WHERE (rowid, c) IN t1_type(func(), 1) ORDER BY a;
 a
---
 1
(1 row)
ora=# -- IN/not IN
ora=# EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid IN func();
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.rowid
   Node/s: All datanodes
   Remote query: SELECT rowid, tableoid, rowno FROM ONLY public.t1 WHERE true
   Coordinator quals: ((t1.rowid = func()) AND ((-1)::smallint = rowid_hashbucketid(func())) AND
(t1.tableoid = rowid_tableoid(func())) AND (t1.rowno = rowid_sequence(func())))
```

```
(5 rows)
ora=# SELECT  rowid FROM t1 WHERE rowid IN func();
            rowid
------------------------------
 AAAAAAAAAP//AAAECpAAAAAAAAAAAB
(1 row)
ora=# EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid IN (func(), func()); -- rowexpr is
not rewritten.
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                         QUERY PLAN
------------------------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.rowid
   Node/s: All datanodes
   Remote query: SELECT rowid FROM ONLY public.t1 WHERE true
   Coordinator quals: (t1.rowid = ANY (ARRAY[func(), func()]))
(5 rows)
ora=# SELECT  rowid FROM t1 WHERE rowid IN (func(), func());
            rowid
------------------------------
 AAAAAAAAAP//AAAECpAAAAAAAAAAAB
(1 row)
ora=# EXPLAIN (costs off, verbose) SELECT  rowid FROM t1 WHERE rowid not IN func();
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                       QUERY PLAN
------------------------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.rowid
   Node/s: All datanodes
   Remote query: SELECT rowid FROM ONLY public.t1 WHERE true
   Coordinator quals: (t1.rowid <> func())
(5 rows)
ora=# SELECT rowid FROM t1 WHERE rowid not IN func();
            rowid
------------------------------
 AAAAAAAAAP//AAAECpAAAAAAAAAAAC
 AAAAAAAAAP//AAAECpAAAAAAAAAAAD
 AAAAAAAAAP//AAAECpAAAAAAAAAAAE
 AAAAAAAAAP//AAAECpAAAAAAAAAAAF
 AAAAAAAAAP//AAAECpAAAAAAAAAAAG
 AAAAAAAABP//AAAECpAAAAAAAAAAAL
 AAAAAAAABP//AAAECpAAAAAAAAAAAM
 AAAAAAAABP//AAAECpAAAAAAAAAAAN
 AAAAAAAABP//AAAECpAAAAAAAAAAAO
(9 rows)
ora=# -- distinct FROM is not rewritten.
ora=# EXPLAIN (costs off, verbose) SELECT a FROM t1 WHERE rowid is distinct FROM func();
WARNING:  Statistics IN some tables OR columns(public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                         QUERY PLAN
------------------------------------------------------------------
 Data Node Scan ON t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.a
   Node/s: All datanodes
   Remote query: SELECT a, rowid FROM ONLY public.t1 WHERE true
   Coordinator quals: (t1.rowid IS DISTINCT FROM func())
(5 rows)
ora=# SELECT a FROM t1 WHERE rowid is distinct FROM func() ORDER BY a;
 a
---
 1
 1
 2
 2
 2
 3
 3
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 3
 3
(9 rows)
-- Predicate condition rewriting in the JOIN ON clause.
ora=# CREATE TABLE t2(a int, c text) WITH (storage_type=astore, hasrowid=on);
CREATE TABLE
ora=#
ora=# INSERT INTO t2 VALUES( 1, func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES( 2,  func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES(3, func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES(3, func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES(1,  func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES( 2,  func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES(3,  func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES( 3,  func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES( 1, func());
INSERT 0 1
ora=# INSERT INTO t2 VALUES( 2, func());
INSERT 0 1
ora=# EXPLAIN (costs off, verbose) SELECT *,t1.rowid, t2.rowid FROM t1 INNER JOIN t2 ON t1.rowid = t2.c;
WARNING:  Statistics IN some tables OR columns(public.t1.a, public.t2.a, public.t2.c) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
----------------------------
 Streaming (TYPE: GATHER)
   Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
   Node/s: All datanodes
   ->  Nested Loop
         Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
         Inner Unique: true
         ->  Streaming(TYPE: BROADCAST)
               Output: t2.a, t2.c, t2.rowid
               Spawn ON: All datanodes
               Consumer Nodes: All datanodes
               ->  Seq Scan ON public.t2
                     Output: t2.a, t2.c, t2.rowid
                     Distribute Key: t2.a
                     Filter: ((-1)::smallint = rowid_hashbucketid((t2.c)::rowid))
         ->  Index Scan using t1_rowno_idx ON public.t1
               Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
               Distribute Key: t1.a
               Index Cond: (t1.rowno = rowid_sequence((t2.c)::rowid))
               Filter: (((-1)::smallint = rowid_hashbucketid(t1.rowid)) AND ((t2.c)::rowid = t1.rowid) AND
(rowid_tableoid((t2.c)::rowid) = t1.tableoid))
(19 rows)
ora=# SELECT *,t1.rowid, t2.rowid FROM t1 INNER JOIN t2 ON t1.rowid = t2.c;
 a | c | a |               c                |             rowid              |             rowid
---+---+---+--------------------------------+--------------------------------+--------------------------------
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAL
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAM
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAN
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAO
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAB
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAC
```

```
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAD
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAE
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAF
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAG
(10 rows)
ora=#
ora=# EXPLAIN (costs off, verbose) SELECT *,t1.rowid, t2.rowid FROM t1 LEFT JOIN t2 ON t1.rowid = t2.c;
WARNING:  Statistics IN some tables OR columns(public.t1.a, public.t2.a, public.t2.c) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
----------------------
 Streaming (TYPE: GATHER)
   Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
   Node/s: All datanodes
   ->  Nested Loop Left Join
         Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
         Join Filter: ((t1.rowid = (t2.c)::rowid) AND (t1.tableoid = rowid_tableoid((t2.c)::rowid)) AND (t1.rowno
= rowid_sequence((t2.c)::rowid)))
         ->  Index Scan using t1_rowno_idx ON public.t1
               Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
               Distribute Key: t1.a
         ->  Materialize
               Output: t2.a, t2.c, t2.rowid
               ->  Streaming(TYPE: BROADCAST)
                     Output: t2.a, t2.c, t2.rowid
                     Spawn ON: All datanodes
                     Consumer Nodes: All datanodes
                     ->  Seq Scan ON public.t2
                           Output: t2.a, t2.c, t2.rowid
                           Distribute Key: t2.a
                           Filter: ((-1)::smallint = rowid_hashbucketid((t2.c)::rowid))
(19 rows)
ora=# SELECT *,t1.rowid, t2.rowid FROM t1 LEFT JOIN t2 ON t1.rowid = t2.c;
 a | c  | a |               c               |             rowid             |             rowid
---+----+---+-------------------------------+-------------------------------+-------------------------------
 3 |  3 |   |                               | AAAAAAAABP//AAAECpAAAAAAAAAAL |
 3 |  4 |   |                               | AAAAAAAABP//AAAECpAAAAAAAAAAM |
 3 |  7 |   |                               | AAAAAAAABP//AAAECpAAAAAAAAAAN |
 3 |  8 |   |                               | AAAAAAAABP//AAAECpAAAAAAAAAAO |
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAB
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAC
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAD
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAE
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAF
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAG
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAL
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAM
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAN
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAO
 2 |  2 |   |                               | AAAAAAAAAP//AAAECpAAAAAAAAAAC |
 1 |  5 |   |                               | AAAAAAAAAP//AAAECpAAAAAAAAAAD |
 2 |  6 |   |                               | AAAAAAAAAP//AAAECpAAAAAAAAAAE |
 1 |  9 |   |                               | AAAAAAAAAP//AAAECpAAAAAAAAAAF |
 2 | 10 |   |                               | AAAAAAAAAP//AAAECpAAAAAAAAAAG |
```

```
(19 rows)
ora=# EXPLAIN (costs off, verbose) SELECT *,t1.rowid, t2.rowid FROM t1 RIGHT JOIN t2 ON t1.rowid = t2.c;
WARNING:  Statistics IN some tables OR columns(public.t2.a, public.t2.c, public.t1.a) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                               QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
--------------------
 Streaming (TYPE: GATHER)
   Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
   Node/s: All datanodes
   ->  Hash Right Join
         Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
         Hash Cond: ((t1.rowid = (t2.c)::rowid) AND (t1.tableoid = rowid_tableoid((t2.c)::rowid)) AND
(t1.rowno = rowid_sequence((t2.c)::rowid)))
         Join Filter: ((-1)::smallint = rowid_hashbucketid((t2.c)::rowid))
         ->  Streaming(TYPE: BROADCAST)
               Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
               Spawn ON: All datanodes
               Consumer Nodes: All datanodes
               ->  Index Scan using t1_rowno_idx ON public.t1
                     Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
                     Distribute Key: t1.a
         ->  Hash
               Output: t2.a, t2.c, t2.rowid
               ->  Seq Scan ON public.t2
                     Output: t2.a, t2.c, t2.rowid
                     Distribute Key: t2.a
(19 rows)
ora=# SELECT *,t1.rowid, t2.rowid FROM t1 RIGHT JOIN t2 ON t1.rowid = t2.c;
 a | c | a |               c               |             rowid             |             rowid
---+---+---+-------------------------------+-------------------------------+-------------------------------
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAG
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAF
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAE
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAD
 1 | 1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAC
 1 | 1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAB
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAO
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAN
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAM
 1 | 1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAABP//
AAAECzAAAAAAAAAAL
(10 rows)
ora=#
ora=# EXPLAIN (costs off, verbose) SELECT *,t1.rowid, t2.rowid FROM t1 FULL JOIN t2 ON t1.rowid = t2.c;
WARNING:  Statistics IN some tables OR columns(public.t1.a, public.t2.a, public.t2.c) are not collected.
HINT:  Do analyze for them IN ORDER to generate optimized plan.
                                                               QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
--------------------
 Streaming (TYPE: GATHER)
   Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
   Node/s: All datanodes
   ->  Hash Full Join
         Output: t1.a, t1.c, t2.a, t2.c, t1.rowid, t2.rowid
         Inner Unique: true
         Hash Cond: (((t2.c)::rowid = t1.rowid) AND (rowid_tableoid((t2.c)::rowid) = t1.tableoid) AND
(rowid_sequence((t2.c)::rowid) = t1.rowno))
         Join Filter: ((-1)::smallint = rowid_hashbucketid((t2.c)::rowid))
         ->  Streaming(TYPE: REDISTRIBUTE)
```

```
            Output: t2.a, t2.c, t2.rowid, (rowid_sequence((t2.c)::rowid))
            Distribute Key: (rowid_sequence((t2.c)::rowid))
            Spawn ON: All datanodes
            Consumer Nodes: All datanodes
            ->  Seq Scan ON public.t2
                  Output: t2.a, t2.c, t2.rowid, rowid_sequence((t2.c)::rowid)
                  Distribute Key: t2.a
      ->  Hash
            Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
            ->  Streaming(TYPE: REDISTRIBUTE)
                  Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
                  Distribute Key: t1.rowno
                  Spawn ON: All datanodes
                  Consumer Nodes: All datanodes
                  ->  Index Scan using t1_rowno_idx ON public.t1
                        Output: t1.a, t1.c, t1.rowid, t1.tableoid, t1.rowno
                        Distribute Key: t1.a
(26 rows)
ora=# SELECT *,t1.rowid, t2.rowid FROM t1 FULL JOIN t2 ON t1.rowid = t2.c;
 a | c  | a |              c               |            rowid             |             rowid
---+----+---+------------------------------+------------------------------+-------------------------------
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAB
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAC
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAD
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAE
 1 |  1 | 1 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAF
 1 |  1 | 2 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAAAP//AAAECzAAAAAAAAAAG
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAL
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAM
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAN
 1 |  1 | 3 | AAAAAAAAAP//AAAECpAAAAAAAAAAB | AAAAAAAAAP//AAAECpAAAAAAAAAAB |
AAAAAAAABP//AAAECzAAAAAAAAAAO
 1 |  9 |   |                              | AAAAAAAAAP//AAAECpAAAAAAAAAAF |
 2 |  2 |   |                              | AAAAAAAAAP//AAAECpAAAAAAAAAAC |
 3 |  7 |   |                              | AAAAAAAABP//AAAECpAAAAAAAAAAN |
 3 |  4 |   |                              | AAAAAAAABP//AAAECpAAAAAAAAAAM |
 2 | 10 |   |                              | AAAAAAAAAP//AAAECpAAAAAAAAAAG |
 2 |  6 |   |                              | AAAAAAAAAP//AAAECpAAAAAAAAAAE |
 3 |  3 |   |                              | AAAAAAAABP//AAAECpAAAAAAAAAAL |
 3 |  8 |   |                              | AAAAAAAABP//AAAECpAAAAAAAAAAO |
 1 |  5 |   |                              | AAAAAAAAAP//AAAECpAAAAAAAAAAD |
(19 rows)

ora=# -- Single-node table.
ora=# CREATE TABLE t1_single(a int, c int) WITH (storage_type=astore, hasrowid=on) to group
sys_single_group1;
CREATE TABLE
ora=# INSERT INTO t1_single VALUES(1, 1);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(2, 2);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(3, 3);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(3, 4);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(1, 5);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(2, 6);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(3, 7);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(3, 8);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(1, 9);
INSERT 0 1
ora=# INSERT INTO t1_single VALUES(2, 10);
INSERT 0 1
-- WHERE clause rewriting.
ora=# EXPLAIN (costs off, verbose) SELECT * FROM t1_single WHERE rowid = func();
WARNING:  Statistics in some tables or columns(public.t1_single) are not collected.
HINT:  Do analyze for them in order to generate optimized plan.
                                                                                                 QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
 Data Node Scan on t1_single "_REMOTE_TABLE_QUERY_"
   Output: t1_single.a, t1_single.c
   Node/s: (sys_single_group1) datanode1
   Remote query: SELECT a, c, rowid, tableoid, rowno FROM ONLY public.t1_single WHERE true
   Coordinator quals: ((t1_single.rowid = func()) AND ((-1)::smallint = rowid_hashbucketid(func())) AND
(t1_sin
gle.tableoid = rowid_tableoid(func())) AND (t1_single.rowno = rowid_sequence(func())))
(5 rows)

-- Rewriting for single-node tables in non-equi scenarios.
ora=# SET max_datanode_for_plan = 1;
SET
ora=# EXPLAIN verbose SELECT * FROM t1_single WHERE rowid > 'AAAAAAAAAP//AAAI/TAAAAAAAAAAB';
                                      QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Output: t1_single.a, t1_single.c
   Node/s: datanode1
   Remote query: SELECT a, c FROM public.t1_single WHERE rowid > 'AAAAAAAAAP//AAAI/
TAAAAAAAAAAB'::rowid

 Remote SQL: SELECT a, c FROM public.t1_single WHERE rowid > 'AAAAAAAAAP//AAAI/
TAAAAAAAAAAB'::rowid
 Datanode Name: datanode1
   [Bypass]
   Index Scan using t1_single_rowno_idx on public.t1_single  (cost=0.00..24.88 rows=716 width=8)
     Output: a, c
     Index Cond: (t1_single.rowno > 1::bigint)

(12 rows)


-- Clean up data.
ora=# DROP TABLE t1;
DROP TABLE
ora=# DROP TABLE t2;
DROP TABLE
ora=# DROP TABLE t1_single;
DROP TABLE
ora=# DROP FUNCTION func;
DROP FUNCTION
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

ORDER BY Statement Rewriting for the System Column ROWID

If the "ORDER BY rowid" statement is delivered to DNs in distributed mode, it can be rewritten as "ORDER BY tableoid, rowno". If the table is an ordinary table, a single-node table, or pruned to a single partition, the statement can be further simplified to "ORDER BY rowno". When sorting the system column **rowid**, the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

system sequentially compares its constituent fields: **node_slice**, **table_oid**, **bucket_id**, and **row_no**. However, no index is created for the system column **rowid**, whereas a unique index is created for the **rowno** column. Therefore, in cases where the table is an ordinary table, a single-node table, or pruned to a single partition, "ORDER BY rowid" is functionally equivalent to "ORDER BY rowno". Since the **rowno** column has an index, using "ORDER BY rowno" is more efficient. When their sorting costs are equal, "ORDER BY rowid" is rewritten as "ORDER BY rowno".

NOTE

If the "ORDER BY rowid" statement is not pushed down to DNs for execution, rewriting is not supported. If the SQL statement is pushed down to DNs, the following specifications and limitations apply:

- When "ORDER BY rowid" is executed on a single column in an ordinary table or a single-node table, it is rewritten as "ORDER BY rowno".
- When "ORDER BY rowid" is executed on a single column in a partitioned table or single-node partitioned table, it is rewritten as "ORDER BY tableoid, rowno". If the table is pruned to a single partition, the statement is further simplified to "ORDER BY rowno".
- When "ORDER BY" is executed on multiple columns—one of which is the system column **tableoid**—in an ordinary table, a single-node table, or a partitioned table pruned to a single partition, the redundant "ORDER BY tableoid" clause can be removed. For details, refer to the description below.
- When executed on multiple columns, "ORDER BY rowid" cannot be rewritten.
- When the statement contains GROUP or DISTINCT, "ORDER BY rowid" cannot be rewritten.
- When used within aggregate functions, "ORDER BY rowid" cannot be rewritten.
- Rewriting is supported only when the base table a physical table. If the base table is a subquery or CTE, rewriting is not supported.

In addition to these "ORDER BY rowid" rewriting scenarios, redundant **tableoid** conditions (including user-added **tableoid** judgment conditions) can be eliminated. Specifically, **tableoid** sorting can be removed when the following conditions are met:

- There are two or more sorting keys.
- If the SQL statement is not pushed down to DNs, only these tables are supported: ordinary replication tables, single-node tables, and partitioned replication tables and single-node partitioned tables that can be pruned to a single partition. If the SQL statement is pushed down to DNs, only ordinary tables, single-node tables, and partitioned tables that can be pruned to a single partition are supported.

Example:

The value of the **tableoid** field may vary across scenarios. The following example is provided for reference only. Please refer to the actual output in case of any differences.

Assuming that the GUC parameter **explain_perf_mode** is set to **normal**:

```
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".
-- Ordinary table.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# SET max_datanode_for_plan = 64;
SET
ora=# SET cursor_sharing = force;
SET
ora=# SET dist_query_adaptive_mode=off;
SET
ora=# CREATE TABLE t2 (
ora(#     c1 INTEGER,
ora(#     c2 INTEGER,
ora(# c3 INTEGER
ora(# ) WITH (storage_type = astore, hasrowid);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' AS the distribution column BY default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t2 VALUES(generate_series(1,100),generate_series(1,100),generate_series(1,300));
INSERT 0 300
ora=# ANALYZE t2;
ANALYZE
ora=# SET enable_stream_operator=off;
SET
-- light proxy
ora=# SET enable_fast_query_shipping = on;
SET
ora=# SET enable_light_proxy = on;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ * FROM t2 WHERE c1 = 100 ORDER
BY rowid;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t2.c1, t2.c2, t2.c3
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t2 WHERE c1 = $1 ORDER BY rowid


 Remote SQL: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t2 WHERE c1 = 100 ORDER BY rowid
 Datanode Name: datanode2
   [Parameterized]
   Index Scan using t2_rowno_idx on public.t2
     Output: c1, c2, c3, rowid, tableoid, rowno
     Filter: (t2.c1 = $1)
(13 rows)
ora=# SELECT /*+ indexscan(t2) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2 WHERE c1 =
100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
-- FQS
ora=# SET enable_fast_query_shipping = on;
SET
ora=# SET enable_light_proxy = off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ * FROM t2 WHERE c1 = 100 ORDER
BY rowid;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t2.c1, t2.c2, t2.c3
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t2 WHERE c1 = $1 ORDER BY rowid


 Remote SQL: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t2 WHERE c1 = 100 ORDER BY rowid
 Datanode Name: datanode2
   [Parameterized]
   Index Scan using t2_rowno_idx on public.t2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    Output: c1, c2, c3, rowid, tableoid, rowno
    Filter: (t2.c1 = $1)
(13 rows)
ora=# SELECT /*+ indexscan(t2) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2 WHERE c1 =
100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
-- stream
ora=# RESET enable_stream_operator;
RESET
ora=# SET enable_light_proxy = off;
SET
ora=# SET enable_fast_query_shipping = off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ * FROM t2 ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2)
                QUERY PLAN
------------------------------------------
 [Parameterized]
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Merge Sort Key: t2.rowid
   Node/s: All datanodes
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2.rowid
         ->  Seq Scan on public.t2
               Output: c1, c2, c3, rowid
               Distribute Key: c1
(11 rows)
ora=# SELECT /*+ indexscan(t2) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2 ORDER BY
rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+----
            0 |              1 |  1
(1 row)
ora=#
ora=# RESET enable_light_proxy;
RESET
ora=# RESET enable_stream_operator;
RESET
ora=# RESET enable_fast_query_shipping;
RESET


-- Partitioned table.
ora=# CREATE TABLE t2_p (
ora(#     c1 INTEGER,
ora(#     c2 INTEGER,
ora(# c3 INTEGER
ora(# ) WITH (storage_type = astore, hasrowid)
ora-# PARTITION BY RANGE (c2)
ora-# (
ora(#     PARTITION p2023q1 VALUES LESS THAN (100),
ora(#     PARTITION p2023q2 VALUES LESS THAN (200),
ora(# PARTITION p2023q3 VALUES LESS THAN (MAXVALUE)
ora(# );
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' AS the distribution column BY default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t2_p VALUES(generate_series(1,200),generate_series(1,300),generate_series(1,600));
INSERT 0 600
ora=# ANALYZE t2_p;
ANALYZE
ora=#
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2_p)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                    QUERY PLAN
-----------------------------------------------------------
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Merge Sort Key: t2_p.rowid
   Node/s: All datanodes
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2_p.rowid
         ->  Partition Iterator
               Output: c1, c2, c3, rowid
               Iterations: 3
               ->  Partitioned Seq Scan on public.t2_p
                     Output: c1, c2, c3, rowid
                     Distribute Key: c1
                     Selected Partitions:  1..3
(14 rows)
ora=# SET enable_stream_operator=off;
SET
-- light proxy
ora=# SET enable_fast_query_shipping = on;
SET
ora=# SET enable_light_proxy = on;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c1=100
ORDER BY rowid;
                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t2_p.c1, t2_p.c2, t2_p.c3
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3 FROM public.t2_p WHERE c1 = $1 ORDER BY rowid


 Remote SQL: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3 FROM public.t2_p WHERE c1 = 100 ORDER BY rowid
 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: c1, c2, c3, rowid, tableoid, rowno
     Sort Key: t2_p.tableoid, t2_p.rowno
     ->  Partition Iterator
           Output: c1, c2, c3, rowid, tableoid, rowno
           Iterations: 3
           ->  Partitioned Seq Scan on public.t2_p
                 Output: c1, c2, c3, rowid, tableoid, rowno
                 Filter: (t2_p.c1 = $1)
                 Selected Partitions:  1..3




(20 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p WHERE
c1=100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
-- FQS
ora=# SET enable_fast_query_shipping = on;
SET
ora=# SET enable_light_proxy = off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p ORDER BY rowid;--
pgxc
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 [Parameterized]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Sort
   Output: t2_p.c1, t2_p.c2, t2_p.c3, t2_p.rowid
   Sort Key: t2_p.rowid
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"
         Output: t2_p.c1, t2_p.c2, t2_p.c3, t2_p.rowid
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3, rowid FROM ONLY public.t2_p WHERE true
ORDER BY 4


 Remote SQL: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3, rowid FROM ONLY public.t2_p WHERE true ORDER
BY 4
 Datanode Name: datanode1
   [Parameterized]
   Sort
     Output: c1, c2, c3, rowid, tableoid, rowno
     Sort Key: t2_p.tableoid, t2_p.rowno
     ->  Result
           Output: c1, c2, c3, rowid, tableoid, rowno
           One-Time Filter: $1
           ->  Partition Iterator
                 Output: c1, c2, c3, rowid, tableoid, rowno
                 Iterations: 3
                 ->  Partitioned Seq Scan on public.t2_p
                       Output: c1, c2, c3, rowid, tableoid, rowno
                       Selected Partitions:  1..3


 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: c1, c2, c3, rowid, tableoid, rowno
     Sort Key: t2_p.tableoid, t2_p.rowno
     ->  Result
           Output: c1, c2, c3, rowid, tableoid, rowno
           One-Time Filter: $1
           ->  Partition Iterator
                 Output: c1, c2, c3, rowid, tableoid, rowno
                 Iterations: 3
                 ->  Partitioned Seq Scan on public.t2_p
                       Output: c1, c2, c3, rowid, tableoid, rowno
                       Selected Partitions:  1..3
(40 rows)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c1=100
ORDER BY rowid;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t2_p.c1, t2_p.c2, t2_p.c3
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3 FROM public.t2_p WHERE c1 = $1 ORDER BY rowid


 Remote SQL: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3 FROM public.t2_p WHERE c1 = 100 ORDER BY rowid
 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: c1, c2, c3, rowid, tableoid, rowno
     Sort Key: t2_p.tableoid, t2_p.rowno
     ->  Partition Iterator
           Output: c1, c2, c3, rowid, tableoid, rowno
           Iterations: 3
           ->  Partitioned Seq Scan on public.t2_p
                 Output: c1, c2, c3, rowid, tableoid, rowno
                 Filter: (t2_p.c1 = $1)
                 Selected Partitions:  1..3
(20 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p WHERE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
c1=100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
-- stream
ora=# RESET enable_stream_operator;
RESET
ora=# SET enable_light_proxy = off;
SET
ora=# SET enable_fast_query_shipping = off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2_p)
                     QUERY PLAN
-----------------------------------------------------------
 [Parameterized]
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Merge Sort Key: t2_p.rowid
   Node/s: All datanodes
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2_p.rowid
         ->  Partition Iterator
               Output: c1, c2, c3, rowid
               Iterations: 3
               ->  Partitioned Seq Scan on public.t2_p
                     Output: c1, c2, c3, rowid
                     Distribute Key: c1
                     Selected Partitions:  1..3
(15 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p ORDER BY
rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+----
            0 |              1 |  1
(1 row)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c1=100
ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2_p)
                     QUERY PLAN
-----------------------------------------------------------
 [Parameterized]
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Node/s: datanode2
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2_p.rowid
         ->  Partition Iterator
               Output: c1, c2, c3, rowid
               Iterations: 3
               ->  Partitioned Seq Scan on public.t2_p
                     Output: c1, c2, c3, rowid
                     Distribute Key: c1
                     Filter: (t2_p.c1 = 100)
                     Selected Partitions:  1..3
(15 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p WHERE
c1=100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c2=100
ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2_p)
                   QUERY PLAN
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-------------------------------------------------
 [Parameterized]
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Merge Sort Key: t2_p.rowid
   Node/s: All datanodes
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2_p.rowid
         ->  Partitioned Seq Scan on public.t2_p
               Output: c1, c2, c3, rowid
               Distribute Key: c1
               Filter: (t2_p.c2 = 100)
               Selected Partitions:  2
(13 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p WHERE
c2=100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            0 |            402 | 400
(1 row)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c1=100 and
c2=100 ORDER BY rowid;
WARNING:  unused hint: IndexScan(t2_p)
                        QUERY PLAN
-------------------------------------------------------------
 [Parameterized]
 Streaming (type: GATHER)
   Output: c1, c2, c3, rowid
   Node/s: datanode2
   ->  Sort
         Output: c1, c2, c3, rowid
         Sort Key: t2_p.rowid
         ->  Partitioned Seq Scan on public.t2_p
               Output: c1, c2, c3, rowid
               Distribute Key: c1
               Filter: ((t2_p.c1 = 100) AND (t2_p.c2 = 100))
               Selected Partitions:  2
(12 rows)
ora=# SELECT /*+ indexscan(t2_p) */ rowid_nodeid(rowid),rowid_sequence(rowid),c3 FROM t2_p WHERE
c1=100 and c2=100 ORDER BY rowid LIMIT 1;
 rowid_nodeid | rowid_sequence | c3
--------------+----------------+-----
            1 |            108 | 100
(1 row)
ora=# RESET enable_light_proxy;
RESET
ora=# RESET enable_stream_operator;
RESET
ora=# RESET enable_fast_query_shipping;
RESET
-- Redundant table_oid sorting is removed.
ora=# SET enable_stream_operator=off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ rowid,rowno FROM t2 ORDER BY
tableoid,c1;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 [Parameterized]
 Sort
   Output: t2.rowid, t2.rowno, t2.tableoid, t2.c1
   Sort Key: t2.tableoid, t2.c1
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"
         Output: t2.rowid, t2.rowno, t2.tableoid, t2.c1
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t2)*/ rowid, rowno, tableoid, c1 FROM ONLY public.t2 WHERE
true ORDER BY 3, 4
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Remote SQL: SELECT/*+ IndexScan(t2)*/ rowid, rowno, tableoid, c1 FROM ONLY public.t2 WHERE true
ORDER BY 3, 4
 Datanode Name: datanode1
   [Parameterized]
   Sort
     Output: rowid, rowno, tableoid, c1
     Sort Key: t2.c1
     ->  Result
           Output: rowid, rowno, tableoid, c1
           One-Time Filter: $1
           ->  Seq Scan on public.t2
                 Output: rowid, rowno, tableoid, c1


 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: rowid, rowno, tableoid, c1
     Sort Key: t2.c1
     ->  Result
           Output: rowid, rowno, tableoid, c1
           One-Time Filter: $1
           ->  Seq Scan on public.t2
                 Output: rowid, rowno, tableoid, c1
(32 rows)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2_p) */ * FROM t2_p WHERE c2 = 90
ORDER BY tableoid,c1;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 [Parameterized]
 Sort
   Output: t2_p.c1, t2_p.c2, t2_p.c3, t2_p.tableoid
   Sort Key: t2_p.tableoid, t2_p.c1
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"
         Output: t2_p.c1, t2_p.c2, t2_p.c3, t2_p.tableoid
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3, tableoid FROM ONLY public.t2_p WHERE c2 =
90 ORDER BY 4, 1


 Remote SQL: SELECT/*+ IndexScan(t2_p)*/ c1, c2, c3, tableoid FROM ONLY public.t2_p WHERE c2 = 90
ORDER BY 4, 1
 Datanode Name: datanode1
   [Parameterized]
   Sort
     Output: c1, c2, c3, tableoid
     Sort Key: t2_p.c1
     ->  Partitioned Seq Scan on public.t2_p
           Output: c1, c2, c3, tableoid
           Filter: (t2_p.c2 = $1)
           Selected Partitions:  1 (pbe-pruning)


 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: c1, c2, c3, tableoid
     Sort Key: t2_p.c1
     ->  Partitioned Seq Scan on public.t2_p
           Output: c1, c2, c3, tableoid
           Filter: (t2_p.c2 = $1)
           Selected Partitions:  1 (pbe-pruning)



(30 rows)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT * FROM (SELECT /*+ indexscan(t2) */ rowid,* FROM t2
ORDER BY c2,tableoid);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
--
 [Parameterized]
 Subquery Scan on __unnamed_subquery__
   Output: __unnamed_subquery__.rowid, __unnamed_subquery__.c1, __unnamed_subquery__.c2,
__unnamed_subquery__.c3
   ->  Sort
         Output: t2.rowid, t2.c1, t2.c2, t2.c3, t2.tableoid
         Sort Key: t2.c2, t2.tableoid
         ->  Data Node Scan on "__REMOTE_SORT_QUERY__"
               Output: t2.rowid, t2.c1, t2.c2, t2.c3, t2.tableoid
               Node/s: All datanodes
               Remote query: SELECT/*+ IndexScan(t2)*/ rowid, c1, c2, c3, tableoid FROM ONLY public.t2
WHERE true ORDER BY 3, 5


 Remote SQL: SELECT/*+ IndexScan(t2)*/ rowid, c1, c2, c3, tableoid FROM ONLY public.t2 WHERE true
ORDER BY 3, 5
 Datanode Name: datanode1
   [Parameterized]
   Sort
     Output: rowid, c1, c2, c3, tableoid
     Sort Key: t2.c2
     ->  Result
           Output: rowid, c1, c2, c3, tableoid
           One-Time Filter: $1
           ->  Seq Scan on public.t2
                 Output: rowid, c1, c2, c3, tableoid


 Datanode Name: datanode2
   [Parameterized]
   Sort
     Output: rowid, c1, c2, c3, tableoid
     Sort Key: t2.c2
     ->  Result
           Output: rowid, c1, c2, c3, tableoid
           One-Time Filter: $1
           ->  Seq Scan on public.t2
                 Output: rowid, c1, c2, c3, tableoid
(34 rows)
-- In replication table scenarios where ORDER BY is not pushed down to DNs, tableoid sorting is removed.
ora=# CREATE TABLE t1 (
ora(#     c1 INTEGER,
ora(#     c2 INTEGER,
ora(#     c3 INTEGER
ora(# ) WITH (storage_type = astore) DISTRIBUTE BY REPLICATION;
CREATE TABLE
ora=#
ora=# INSERT INTO t1 VALUES(generate_series(1,300),generate_series(1,300),generate_series(1,300));
INSERT 0 300
ora=# CREATE OR REPLACE FUNCTION func RETURN integer VOLATILE AS
ora$# BEGIN
ora$#     RETURN 1;
ora$# END;
ora$# /
CREATE FUNCTION
ora=# SET enable_stream_operator=off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT tableoid,func() FROM t1 ORDER BY tableoid, c1;
WARNING:  Statistics in some tables OR columns(public.t1) are not collected.
HINT:  Do ANALYZE for them in ORDER to generate optimized plan.
                              QUERY PLAN
--------------------------------------------------------------------------------
 Sort
   Output: t1.tableoid, (func()), t1.c1
   Sort Key: t1.c1
   ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
```

```
        Output: t1.tableoid, func(), t1.c1
        Node/s: datanode1
        Remote query: SELECT tableoid, c1 FROM ONLY public.t1 WHERE true


 Remote SQL: SELECT tableoid, c1 FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: tableoid, c1
(13 rows)
ora=# RESET enable_stream_operator;
RESET

-- Single-node table.
ora=# CREATE TABLE t1_single (
ora(#     c1 INTEGER,
ora(#     c2 INTEGER,
ora(# c3 INTEGER
ora(# ) WITH (storage_type = astore, hasrowid) to group sys_single_group1;
CREATE TABLE

ora=# EXPLAIN  (verbose, costs false) SELECT /*+ indexscan(t2) */ * FROM t1_single WHERE c1 = 100
ORDER BY rowid;
                                        QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
-----
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t1_single.c1, t1_single.c2, t1_single.c3
   Node/s: datanode1
   Remote query: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t1_single WHERE c1 = 100 ORDER BY
tableoid, r
owno

 Remote SQL: SELECT/*+ IndexScan(t2)*/ c1, c2, c3 FROM public.t1_single WHERE c1 = 100 ORDER BY
tableoid, rowno
 Datanode Name: datanode1
   Sort
     Output: c1, c2, c3, tableoid, rowno
     Sort Key: t1_single.rowno
     ->  Seq Scan on public.t1_single
           Output: c1, c2, c3, tableoid, rowno
           Filter: (t1_single.c1 = 100)

(14 rows)


-- Clean up data.
ora=# DROP TABLE t1;
DROP TABLE
ora=# DROP TABLE t2;
DROP TABLE
ora=# DROP TABLE t2_p;
DROP TABLE
ora=# DROP TABLE t1_single;
DROP TABLE
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

Max/Min Optimization for the System Column ROWID

For ordinary tables, since the mapping from **ROWID** to **rowno** is strictly order-preserving, the min/max aggregate function involving a single **ROWID** column can be optimized by rewriting it as an expression with the system column **rowno**. By leveraging the ordered nature of indexes, this approach optimizes min/max

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

aggregate operations into LIMIT clauses with index scans, thereby reducing data reads and computational overhead. SQL statements like "SELECT max(rowid) FROM t1" follow this rewriting rule:

```
SELECT rowid FROM t1 ORDER BY rowno DESC LIMIT 1
```

NOTE

Max/Min optimization for the system column **ROWID** is subject to the following specifications and limitations:

- For ordinary tables, SQL statements can be rewritten only when they are pushed down to DNs. Additionally, single-node tables support rewriting when the Stream plan is executed. Partitioned tables are not eligible for rewriting.
- When the GUC parameter **enable_indexscan** is disabled or when index scans cannot be performed on the **rowno** column, this optimization does not take effect by default. For details about the **enable_indexscan** parameter, see "Database GUC Parameters > GUC Parameters > Query Planning > Optimizer Method Configuration" in *Reference*.

Example:

The value of the **tableoid** field may vary across scenarios. The following example is provided for reference only. Please refer to the actual output in case of any differences.

Assuming that the GUC parameter **explain_perf_mode** is set to **normal**:

```
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".
-- Prepare data.
ora=# SET max_datanode_for_plan = 64;
SET
ora=# SET cursor_sharing = force;
SET
ora=# SET dist_query_adaptive_mode=off;
SET
ora=# CREATE TABLE t1 (a int, b int) WITH (STORAGE_TYPE=Astore, hasrowid=ON);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t1 VALUES ( generate_series(1,1000), generate_series(3001,4000));
INSERT 0 1000
ora=# INSERT INTO t1 VALUES ( generate_series(1001,2000), generate_series(3001,4000));
INSERT 0 1000
ora=# CREATE INDEX idx_t1_b ON t1(b);
CREATE INDEX
ora=# ANALYZE t1;
ANALYZE
-- Max/Min optimization scenarios.
ora=# SET enable_stream_operator=off;
SET
ora=# SET enable_fast_query_shipping = ON;
SET
ora=# SET enable_light_proxy = ON;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */ max(rowid) FROM t1;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: max((max(t1.rowid)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (max(t1.rowid))
         Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      Remote query: SELECT/*+ IndexScan(t1)*/ max(rowid) FROM ONLY public.t1 WHERE true



Remote SQL: SELECT/*+ IndexScan(t1)*/ max(rowid) FROM ONLY public.t1 WHERE true
Datanode Name: datanode1
  [Parameterized]
  Result
    Output: $0
    InitPlan 1 (returns $0)
      ->  Limit
            Output: public.t1.rowid, public.t1.rowno
            ->  Result
                  Output: public.t1.rowid, public.t1.rowno
                  One-Time Filter: $1
                  ->  Index Scan Backward using t1_rowno_idx ON public.t1
                        Output: public.t1.rowid, public.t1.rowno


Datanode Name: datanode2
  [Parameterized]
  Result
    Output: $0
    InitPlan 1 (returns $0)
      ->  Limit
            Output: public.t1.rowid, public.t1.rowno
            ->  Result
                  Output: public.t1.rowid, public.t1.rowno
                  One-Time Filter: $1
                  ->  Index Scan Backward using t1_rowno_idx ON public.t1
                        Output: public.t1.rowid, public.t1.rowno
(35 rows)
ora=# SELECT /*+ indexscan(t1) */ rowid_sequence(max(rowid)) FROM t1;
 rowid_sequence
----------------
          2004
(1 row)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */ min(rowid) FROM t1;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: min((min(t1.rowid)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (min(t1.rowid))
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t1)*/ min(rowid) FROM ONLY public.t1 WHERE true


Remote SQL: SELECT/*+ IndexScan(t1)*/ min(rowid) FROM ONLY public.t1 WHERE true
Datanode Name: datanode1
  [Parameterized]
  Result
    Output: $0
    InitPlan 1 (returns $0)
      ->  Limit
            Output: public.t1.rowid, public.t1.rowno
            ->  Result
                  Output: public.t1.rowid, public.t1.rowno
                  One-Time Filter: $1
                  ->  Index Scan using t1_rowno_idx ON public.t1
                        Output: public.t1.rowid, public.t1.rowno


Datanode Name: datanode2
  [Parameterized]
  Result
    Output: $0
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   InitPlan 1 (returns $0)
     ->  Limit
           Output: public.t1.rowid, public.t1.rowno
           ->  Result
                 Output: public.t1.rowid, public.t1.rowno
                 One-Time Filter: $1
                 ->  Index Scan using t1_rowno_idx ON public.t1
                       Output: public.t1.rowid, public.t1.rowno
(35 rows)
ora=# SELECT /*+ indexscan(t1) */ rowid_sequence(min(rowid)) FROM t1; -- 1
 rowid_sequence
----------------
              1
(1 row)
-- Optimization is supported in multiple max/min scenarios.
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */ rowid_sequence(max(rowid)),max(b)
FROM t1;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: rowid_sequence(max((max(t1.rowid)))), max((max(t1.b)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (max(t1.rowid)), (max(t1.b))
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t1)*/ max(rowid), max(b) FROM ONLY public.t1 WHERE true


 Remote SQL: SELECT/*+ IndexScan(t1)*/ max(rowid), max(b) FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   [Parameterized]
   Result
     Output: $0, $1
     InitPlan 1 (returns $0)
       ->  Limit
             Output: public.t1.rowid, public.t1.rowno
             ->  Result
                   Output: public.t1.rowid, public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowid, public.t1.rowno
     InitPlan 2 (returns $1)
       ->  Limit
             Output: public.t1.b
             ->  Result
                   Output: public.t1.b
                   One-Time Filter: $1
                   ->  Index Only Scan Backward using idx_t1_b ON public.t1
                         Output: public.t1.b
                         Index Cond: (public.t1.b IS NOT NULL)


 Datanode Name: datanode2
   [Parameterized]
   Result
     Output: $0, $1
     InitPlan 1 (returns $0)
       ->  Limit
             Output: public.t1.rowid, public.t1.rowno
             ->  Result
                   Output: public.t1.rowid, public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowid, public.t1.rowno
     InitPlan 2 (returns $1)
       ->  Limit
             Output: public.t1.b
             ->  Result
                   Output: public.t1.b
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                One-Time Filter: $1
                ->  Index Only Scan Backward using idx_t1_b ON public.t1
                      Output: public.t1.b
                      Index Cond: (public.t1.b IS NOT NULL)
(53 rows)
ora=# SELECT /*+ indexscan(t1) */ rowid_sequence(max(rowid)),max(b) FROM t1;
 rowid_sequence | max
----------------+------
          2004 | 4000
(1 row)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */
rowid_sequence(max(rowid)),max(rowno) FROM t1;
                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: rowid_sequence(max((max(t1.rowid)))), max((max(t1.rowno)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (max(t1.rowid)), (max(t1.rowno))
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t1)*/ max(rowid), max(rowno) FROM ONLY public.t1 WHERE
true


 Remote SQL: SELECT/*+ IndexScan(t1)*/ max(rowid), max(rowno) FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   [Parameterized]
   Result
     Output: $0, $1
     InitPlan 1 (returns $0)
       ->  Limit
             Output: public.t1.rowid, public.t1.rowno
             ->  Result
                   Output: public.t1.rowid, public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowid, public.t1.rowno
     InitPlan 2 (returns $1)
       ->  Limit
             Output: public.t1.rowno
             ->  Result
                   Output: public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Only Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowno


 Datanode Name: datanode2
   [Parameterized]
   Result
     Output: $0, $1
     InitPlan 1 (returns $0)
       ->  Limit
             Output: public.t1.rowid, public.t1.rowno
             ->  Result
                   Output: public.t1.rowid, public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowid, public.t1.rowno
     InitPlan 2 (returns $1)
       ->  Limit
             Output: public.t1.rowno
             ->  Result
                   Output: public.t1.rowno
                   One-Time Filter: $1
                   ->  Index Only Scan Backward using t1_rowno_idx ON public.t1
                         Output: public.t1.rowno
(51 rows)
ora=# SELECT /*+ indexscan(t1) */ rowid_sequence(max(rowid)),max(rowno) FROM t1;
```

```
 rowid_sequence | max
----------------+------
          2004 | 2004
(1 row)
-- When the enable_indexscan parameter is disabled, max/min optimization is not supported.
ora=# SET enable_indexscan = off;
SET
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */ max(rowid) FROM t1;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: max((max(t1.rowid)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (max(t1.rowid))
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t1)*/ max(rowid) FROM ONLY public.t1 WHERE true


 Remote SQL: SELECT/*+ IndexScan(t1)*/ max(rowid) FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   [Parameterized]
   Aggregate
     Output: max(rowid)
     ->  Result
           Output: rowid
           One-Time Filter: $1
           ->  Seq Scan ON public.t1
                 Output: rowid


 Datanode Name: datanode2
   [Parameterized]
   Aggregate
     Output: max(rowid)
     ->  Result
           Output: rowid
           One-Time Filter: $1
           ->  Seq Scan ON public.t1
                 Output: rowid
(29 rows)
ora=# SET enable_indexscan = ON;
SET
-- Max/Min optimization is not supported for complex SQL statements.
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t1) */ rowid_sequence(max(rowid)) FROM
(SELECT rowid,a FROM t1 WHERE a = 1);
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------
 [Parameterized]
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Output: (rowid_sequence(max(__unnamed_subquery__.rowid)))
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t1)*/ rowid_sequence(max(rowid)) AS rowid_sequence FROM
(SELECT t1.rowid, t1.a FROM public.t1 WHERE t1.a = $1) __unnamed_subquery__


 Remote SQL: SELECT/*+ IndexScan(t1)*/ rowid_sequence(max(rowid)) AS rowid_sequence FROM (SELECT
t1.rowid, t1.a FROM public.t1 WHERE t1.a = 1) __unnamed_subquery__
 Datanode Name: datanode1
   [Parameterized]
   Aggregate
     Output: rowid_sequence(max(t1.rowid))
     ->  Seq Scan ON public.t1
           Output: t1.rowid
           Filter: (t1.a = $1)
(15 rows)
-- Max/Min optimization is not supported for partitioned tables.
ora=# CREATE TABLE t2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora-# ( a int,
ora(# b int,
ora(# c int)
ora-# WITH (STORAGE_TYPE=Astore, hasrowid=ON)
ora-# PARTITION BY RANGE (a)
ora-# (
ora(# PARTITION P1 VALUES LESS THAN(100),
ora(# PARTITION P2 VALUES LESS THAN(200),
ora(# PARTITION P3 VALUES LESS THAN(300),
ora(# PARTITION P8 VALUES LESS THAN(MAXVALUE)
ora(# ) ;
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t2 VALUES ( generate_series(1,1000), generate_series(1001,2000));
INSERT 0 1000
ora=# ANALYZE t2;
ANALYZE
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ max(rowid) FROM t2;
                                QUERY PLAN
-----------------------------------------------------------------------------------------------
 [Parameterized]
 Aggregate
   Output: max((max(t2.rowid)))
   ->  Data Node Scan ON "__REMOTE_GROUP_QUERY__"
         Output: (max(t2.rowid))
         Node/s: All datanodes
         Remote query: SELECT/*+ IndexScan(t2)*/ max(rowid) FROM ONLY public.t2 WHERE true


 Remote SQL: SELECT/*+ IndexScan(t2)*/ max(rowid) FROM ONLY public.t2 WHERE true
 Datanode Name: datanode1
   [Parameterized]
   Aggregate
     Output: max(rowid)
     ->  Result
           Output: rowid
           One-Time Filter: $1
           ->  Partition Iterator
                 Output: rowid
                 Iterations: 4
                 ->  Partitioned Seq Scan ON public.t2
                       Output: rowid
                       Selected Partitions:  1..4


 Datanode Name: datanode2
   [Parameterized]
   Aggregate
     Output: max(rowid)
     ->  Result
           Output: rowid
           One-Time Filter: $1
           ->  Partition Iterator
                 Output: rowid
                 Iterations: 4
                 ->  Partitioned Seq Scan ON public.t2
                       Output: rowid
                       Selected Partitions:  1..4
(37 rows)
ora=# EXPLAIN  (VERBOSE, COSTS FALSE) SELECT /*+ indexscan(t2) */ max(rowid) FROM t2 WHERE a = 1;
                                QUERY PLAN
-----------------------------------------------------------------------------------------------
 [Parameterized]
 Data Node Scan ON "__REMOTE_LIGHT_QUERY__"
   Output: (max(t2.rowid))
   Node expr: $1
   Remote query: SELECT/*+ IndexScan(t2)*/ max(rowid) AS max FROM public.t2 WHERE a = $1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Remote SQL: SELECT/*+ IndexScan(t2)*/ max(rowid) AS max FROM public.t2 WHERE a = 1
 Datanode Name: datanode1
   [Parameterized]
   Aggregate
     Output: max(rowid)
     ->  Partitioned Seq Scan ON public.t2
           Output: rowid
           Filter: (t2.a = $1)
           Selected Partitions:  1 (pbe-pruning)
(16 rows)

-- Single-node table.
ora=# CREATE TABLE t1_single (a int, b int) WITH (STORAGE_TYPE=Astore, hasrowid=on) to group
sys_single_group1;
CREATE TABLE
ora=# INSERT INTO t1_single VALUES ( generate_series(1,1000), generate_series(3001,4000));
INSERT 0 1000
ora=# INSERT INTO t1_single VALUES ( generate_series(1001,2000), generate_series(3001,4000));
INSERT 0 1000
ora=# EXPLAIN  (verbose, costs false) SELECT /*+ indexscan(t1_single) */ max(rowid) FROM t1_single;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: (max(t1_single.rowid))
   Node/s: datanode1
   Remote query: SELECT/*+ IndexScan(t1_single)*/ max(rowid) AS max FROM public.t1_single

 Remote SQL: SELECT/*+ IndexScan(t1_single)*/ max(rowid) AS max FROM public.t1_single
 Datanode Name: datanode1
   Result
     Output: $0
     InitPlan 1 (returns $0)
       ->  Limit
             Output: public.t1_single.rowid, public.t1_single.rowno
             ->  Index Scan Backward using t1_single_rowno_idx on public.t1_single
                   Output: public.t1_single.rowid, public.t1_single.rowno

(15 rows)


-- Clean up data.
ora=# DROP TABLE t2;
DROP TABLE
ora=# DROP TABLE t1;
DROP TABLE
ora=# DROP TABLE t1_single;
DROP TABLE
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

Distribution/Partition Pruning for the System Column ROWID

Distribution pruning is triggered when the system column **ROWID** is used as an equality predicate condition. This allows the query to be pruned to a single unique DN. During the distributed static pruning phase, the system performs **ROWID** pruning prior to distribution key pruning. After the predicate (**ROWID** column) is obtained, if the query is confirmed to be prunable to a single DN, the pushdown node is computed directly. If external parameters are involved, the relevant parameters are saved, and the pushdown node is computed during the dynamic pruning phase. For unsupported scenarios, regular distribution key pruning is applied.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

Distribution pruning adheres to the same specifications and limitations as existing distribution key pruning:

- Only **ROWID** can be used as an equality or NULL expression.
- In Boolean combinations, only the first ROWID condition in AND expressions is pruned; OR expressions are not supported.
- Pruning applies only to constants (static pruning) and parameters (dynamic pruning).
- Static pruning supports four types of distributed plans. Dynamic pruning supports only the CN lightweight, FQS, and PGXC plans.
- Distribution pruning does not support functions.
- If type conversion is required for a column or condition used in distribution pruning, only implicit binary forcible conversion is supported. Function-based conversion and explicit conversion are not supported.

Pruning to a single unique partition is allowed when the predicate condition for the system column **ROWID** is rewritten to include **tableoid** equality judgment. Both constant and parameter conditions are supported: Static pruning is triggered for ROWID constant conditions. Dynamic pruning is triggered for ROWID parameter conditions. For details about partition pruning, see **Partition Pruning**.

Examples:

The following is an example of distribution pruning:

The value of the **tableoid** field may vary across scenarios. The following example is provided for reference only. Please refer to the actual output in case of any differences.

Assuming that the GUC parameter **explain_perf_mode** is set to **normal**:

```
-- Prepare data.
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".
ora=# SET max_datanode_for_plan = 1;
SET
ora=# SET query_dop=1;
SET
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# SET auto_explain_level=notice;
SET
ora=# CREATE TABLE t1 (c1 int, c2 int) WITH (storage_type=astore, hasrowid=on);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t1 SELECT generate_series(1,3), generate_series(1,4);
INSERT 0 12
ora=# ANALYZE t1;
ANALYZE
-- CN lightweight plan.
-- Static pruning of constant expressions.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE';
                        QUERY PLAN




-----------------------------------------------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid) AND tableoid = rowid_tab
leoid('AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND rowno = rowid_sequence('AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid)


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid) AND tableoid = rowid_tableoi
d('AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND rowno = rowid_sequence('AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid)
 Datanode Name: datanode1
   Index Scan using t1_rowno_idx on public.t1
     Output: c1, c2
     Index Cond: (t1.rowno = 4::bigint)
     Filter: ((t1.rowid = 'AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND (t1.tableoid = 16654::oid))
(12 rows)
-- Dynamic pruning of parameter expressions.
ora=# SET plan_cache_mode = force_generic_plan;
SET
ora=# PREPARE p1 AS SELECT * FROM t1 WHERE rowid = $1;
PREPARE
ora=# EXPLAIN (costs off, verbose on) EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
                                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Node expr: $1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid = $1 AND (-1)::smallint =
rowid_hashbucketid($1) AND tableoid = rowid_tableoid($1) AND rowno = rowid_sequence($1)


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid = $1 AND (-1)::smallint =
rowid_hashbucketid($1) AND tableoid = rowid_tableoid($1) AND rowno = rowid_sequence($1)
 Datanode Name: datanode1
   Result
     Output: c1, c2
     One-Time Filter: ((-1)::smallint = rowid_hashbucketid($1))
     ->  Index Scan using t1_rowno_idx on public.t1
           Output: c1, c2
           Index Cond: (t1.rowno = rowid_sequence($1))
           Filter: ((t1.rowid = $1) AND (t1.tableoid = rowid_tableoid($1)))
(16 rows)
ora=# EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
 c1 | c2
----+----
  2 |  1
(1 row)
ora=# DEALLOCATE p1;
DEALLOCATE
-- Static pruning is performed for the first ROWID expression in AND expressions.
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
                               QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL AND c1 = 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL AND c1 = 1
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = 1)
(11 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
 c1 | c2
----+----
  1 |  1
  1 |  4
  1 |  3
  1 |  2
(4 rows)
-- OR expressions do not support pruning.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL OR c1 = 1


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL OR c1 = 1
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
(10 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  1 |  4
  2 |  1
  1 |  3
  2 |  4
  1 |  2
  2 |  3
  3 |  3
  3 |  2
  3 |  1
  3 |  4
(12 rows)


-- FQS plan.
ora=# SET enable_light_proxy = off;
SET
ora=# SET plan_cache_mode = force_custom_plan;
SET
-- Static pruning of constant expressions.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE';
                     QUERY PLAN




-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAAAP//
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
AAAEEOAAAAAAAAAAAE'::rowid) AND tableoid = rowid_tab
leoid('AAAAAAAAAP//AAAEEOAAAAAAAAAAAE'::rowid) AND rowno = rowid_sequence('AAAAAAAAAP//
AAAEEOAAAAAAAAAAAE'::rowid)


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAAE'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAAAP//
AAAEEOAAAAAAAAAAAE'::rowid) AND tableoid = rowid_tableoi
d('AAAAAAAAAP//AAAEEOAAAAAAAAAAAE'::rowid) AND rowno = rowid_sequence('AAAAAAAAAP//
AAAEEOAAAAAAAAAAAE'::rowid)
 Datanode Name: datanode1
   Index Scan using t1_rowno_idx on public.t1
     Output: c1, c2
     Index Cond: (t1.rowno = 4::bigint)
     Filter: ((t1.rowid = 'AAAAAAAAAP//AAAEEOAAAAAAAAAAAE'::rowid) AND (t1.tableoid = 16654::oid))
(12 rows)
-- Dynamic pruning of parameter expressions.
ora=# SET plan_cache_mode = force_generic_plan;
SET
ora=# PREPARE p1 AS SELECT * FROM t1 WHERE rowid = $1;
PREPARE
ora=# EXPLAIN (costs off, verbose on) EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAAE');
                                                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Node expr: $1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid = $1 AND (-1)::smallint =
rowid_hashbucketid($1) AND tableoid = rowid_tableoid($1) AND rowno = rowid_sequence($1)


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid = $1 AND (-1)::smallint =
rowid_hashbucketid($1) AND tableoid = rowid_tableoid($1) AND rowno = rowid_sequence($1)
 Datanode Name: datanode1
   Result
     Output: c1, c2
     One-Time Filter: ((-1)::smallint = rowid_hashbucketid($1))
     ->  Index Scan using t1_rowno_idx on public.t1
           Output: c1, c2
           Index Cond: (t1.rowno = rowid_sequence($1))
           Filter: ((t1.rowid = $1) AND (t1.tableoid = rowid_tableoid($1)))
(16 rows)
ora=# EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAAE');
 c1 | c2
----+----
  2 |  1
(1 row)
ora=# DEALLOCATE p1;
DEALLOCATE
-- Static pruning is performed for the first ROWID expression in AND expressions.
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL AND c1 = 1


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL AND c1 = 1
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = 1)
(11 rows)
```

```
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
 c1 | c2
----+----
  1 |  1
  1 |  4
  1 |  3
  1 |  2
(4 rows)
-- OR expressions do not support pruning.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1; --all
                              QUERY PLAN
-------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL OR c1 = 1


 Remote SQL: SELECT c1, c2 FROM public.t1 WHERE rowid IS NOT NULL OR c1 = 1
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
(10 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  1 |  4
  2 |  1
  1 |  3
  2 |  4
  1 |  2
  2 |  3
  3 |  3
  3 |  2
  3 |  1
  3 |  4
(12 rows)


-- Stream plan.
ora=# SET enable_fast_query_shipping = off;
SET
ora=# SET plan_cache_mode = force_custom_plan;
SET
-- Static pruning of constant expressions.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE';
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: c1, c2
   Node/s: datanode1
   ->  Index Scan using t1_rowno_idx on public.t1
         Output: c1, c2
         Distribute Key: c1
         Index Cond: (t1.rowno = 4::bigint)
         Filter: ((t1.rowid = 'AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND (t1.tableoid = 16654::oid))
(8 rows)
-- Parameter expressions. Dynamic pruning does not apply to the Stream plan.
ora=# SET plan_cache_mode = force_generic_plan;
SET
ora=# PREPARE p1 AS SELECT * FROM t1 WHERE rowid = $1;
PREPARE
ora=# EXPLAIN (costs off, verbose on) EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
                              QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   Output: c1, c2
   Node/s: All datanodes
   ->  Result
         Output: c1, c2
         One-Time Filter: ((-1)::smallint = rowid_hashbucketid($1))
         ->  Index Scan using t1_rowno_idx on public.t1
               Output: c1, c2
               Distribute Key: c1
               Index Cond: (t1.rowno = rowid_sequence($1))
               Filter: ((t1.rowid = $1) AND (t1.tableoid = rowid_tableoid($1)))
(11 rows)
ora=# EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
 c1 | c2
----+----
  2 |  1
(1 row)
ora=# DEALLOCATE p1;
DEALLOCATE
-- Static pruning is performed for the first ROWID expression in AND expressions.
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Output: c1, c2
   Node/s: datanode1
   ->  Seq Scan on public.t1
         Output: c1, c2
         Distribute Key: c1
         Filter: (t1.c1 = 1)
(7 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
 c1 | c2
----+----
  1 |  1
  1 |  4
  1 |  3
  1 |  2
(4 rows)
-- OR expressions do not support pruning.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Output: c1, c2
   Node/s: All datanodes
   ->  Seq Scan on public.t1
         Output: c1, c2
         Distribute Key: c1
(6 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  1 |  4
  2 |  1
  1 |  3
  2 |  4
  1 |  2
  2 |  3
  3 |  3
  3 |  2
  3 |  1
  3 |  4
(12 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- PGXC plan.
ora=# SET enable_fast_query_shipping = off;
SET
ora=# SET enable_stream_operator = off;
SET
ora=# SET plan_cache_mode = force_custom_plan;
SET
-- Static pruning of constant expressions.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------
 Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND tableoid = 16654::oid AND rowno = 4::bigint


 Remote SQL: SELECT c1, c2 FROM ONLY public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND tableoid = 16654::oid AND rowno = 4::bigint
 Datanode Name: datanode1
   Index Scan using t1_rowno_idx on public.t1
     Output: c1, c2
     Index Cond: (t1.rowno = 4::bigint)
     Filter: ((t1.rowid = 'AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND (t1.tableoid = 16654::oid))
(12 rows)
-- Dynamic pruning of parameter expressions.
ora=# SET plan_cache_mode = force_generic_plan;
SET
ora=# PREPARE p1 AS SELECT * FROM t1 WHERE rowid = $1;
PREPARE
ora=# EXPLAIN (costs off, verbose on) EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
 Result
   Output: t1.c1, t1.c2
   One-Time Filter: ((-1)::smallint = rowid_hashbucketid($1))
   ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
         Output: t1.c1, t1.c2
         Node expr: $1
         Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE rowid = $1 AND tableoid =
rowid_tableoid($1) AND rowno = rowid_sequence($1) AND (-1)::smallint = rowid_hashbucketid($1)


 Remote SQL: SELECT c1, c2 FROM ONLY public.t1 WHERE rowid = 'AAAAAAAAAP//
AAAEEOAAAAAAAAAAE'::rowid AND tableoid = 16654::oid AND rowno = 4::bigint
 Datanode Name: datanode1
   Index Scan using t1_rowno_idx on public.t1
     Output: c1, c2
     Index Cond: (t1.rowno = 4::bigint)
     Filter: ((t1.rowid = 'AAAAAAAAAP//AAAEEOAAAAAAAAAAE'::rowid) AND (t1.tableoid = 16654::oid))
(15 rows)
ora=# EXECUTE p1('AAAAAAAAAP//AAAEEOAAAAAAAAAAE');
 c1 | c2
----+----
  2 |  1
(1 row)
ora=# DEALLOCATE p1;
DEALLOCATE
-- Static pruning is performed for the first ROWID expression in AND expressions.
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
                          QUERY PLAN
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
------------------------------------------------------------------
 Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.c1, t1.c2
   Node/s: datanode1
   Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE c1 = 1


 Remote SQL: SELECT c1, c2 FROM ONLY public.t1 WHERE c1 = 1
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
     Filter: (t1.c1 = 1)
(11 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL AND c1 = 1;
 c1 | c2
----+----
  1 |  1
  1 |  4
  1 |  3
  1 |  2
(4 rows)
-- OR expressions do not support pruning.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
                        QUERY PLAN
------------------------------------------------------------------
 Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
   Output: t1.c1, t1.c2
   Node/s: All datanodes
   Remote query: SELECT c1, c2 FROM ONLY public.t1 WHERE true


 Remote SQL: SELECT c1, c2 FROM ONLY public.t1 WHERE true
 Datanode Name: datanode1
   Seq Scan on public.t1
     Output: c1, c2
(10 rows)
ora=# SELECT * FROM t1 WHERE rowid IS NOT NULL OR c1 = 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  1 |  4
  2 |  1
  1 |  3
  2 |  4
  1 |  2
  2 |  3
  3 |  3
  3 |  2
  3 |  1
  3 |  4
(12 rows)

-- Clean up data.
ora=# DROP TABLE t1;
DROP TABLE
ora=# RESET max_datanode_for_plan;
RESET
ora=# RESET query_dop;
RESET
ora=# RESET enable_light_proxy;
RESET
ora=# RESET enable_fast_query_shipping;
RESET
ora=# RESET enable_stream_operator;
RESET
ora=# RESET plan_cache_mode;
RESET
ora=# RESET auto_explain_level;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

The following is an example of partition pruning:

The value of the **tableoid** field may vary across scenarios. The following example is provided for reference only. Please refer to the actual output in case of any differences.

```
-- Prepare data.
gaussdb=# CREATE DATABASE ora dbcompatibility 'ORA';
CREATE DATABASE
gaussdb=# \c ora
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora" as user "$user".
ora=# SET max_datanode_for_plan = 1;
SET
ora=# SET query_dop=1;
SET
ora=# SET auto_explain_level=notice;
SET
ora=# CREATE TABLE t_part (c1 int, c2 int) WITH (storage_type=astore, hasrowid=on) PARTITION BY
HASH(c1) PARTITIONS 4;
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
ora=# INSERT INTO t_part SELECT generate_series(1,3), generate_series(1,4);
INSERT 0 12
ora=# ANALYZE t_part;
ANALYZE
-- CN lightweight plan in Cplan scenarios.
ora=# SET plan_cache_mode = force_custom_plan;
SET
-- Prune to partition p1.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO';
                            QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_LIGHT_QUERY__"
   Output: t_part.c1, t_part.c2
   Node/s: datanode2
   Remote query: SELECT c1, c2 FROM public.t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid) AND tableoid = rowid
_tableoid('AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND rowno = rowid_sequence('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid)


 Remote SQL: SELECT c1, c2 FROM public.t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid) AND tableoid = rowid_tab
leoid('AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND rowno = rowid_sequence('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid)
 Datanode Name: datanode2
   Partitioned Index Scan using t_part_rowno_idx on public.t_part
     Output: c1, c2
     Index Cond: (t_part.rowno = 14::bigint)
     Filter: ((t_part.rowid = 'AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND (t_part.tableoid =
16664::oid))
     Selected Partitions:  1
(13 rows)


-- FQS plan in Gplan scenarios.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ora=# SET enable_light_proxy = off;
SET
ora=# SET plan_cache_mode = force_generic_plan;
SET
-- Prune to partition p1.
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO';
                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: t_part.c1, t_part.c2
   Node/s: datanode2
   Remote query: SELECT c1, c2 FROM public.t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid) AND tableoid = rowid
_tableoid('AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND rowno = rowid_sequence('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid)


 Remote SQL: SELECT c1, c2 FROM public.t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid AND (-1)::smallint = rowid_hashbucketid('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid) AND tableoid = rowid_tab
leoid('AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND rowno = rowid_sequence('AAAAAAAABP//
AAAEEYAAAAAAAAAAO'::rowid)
 Datanode Name: datanode2
   Partitioned Index Scan using t_part_rowno_idx on public.t_part
     Output: c1, c2
     Index Cond: (t_part.rowno = 14::bigint)
     Filter: ((t_part.rowid = 'AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND (t_part.tableoid =
16664::oid))
     Selected Partitions:  1
(13 rows)


-- Stream plan in Cplan scenarios.
ora=# SET enable_fast_query_shipping = off;
SET
ora=# SET plan_cache_mode = force_custom_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO';
                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: c1, c2
   Node/s: datanode2
   ->  Partition Iterator
         Output: c1, c2
         Iterations: PART
         ->  Partitioned Index Scan using t_part_rowno_idx on public.t_part
               Output: c1, c2
               Distribute Key: c1
               Index Cond: (t_part.rowno = 14::bigint)
               Filter: ((t_part.rowid = 'AAAAAAAABP//AAAEEYAAAAAAAAAAO'::rowid) AND (t_part.tableoid =
16664::oid))
               Selected Partitions:  1..4 (pbe-pruning)
(12 rows)
-- Stream plan in Gplan scenarios.
ora=# SET plan_cache_mode = force_generic_plan;
SET
ora=# EXPLAIN (costs off, verbose on) SELECT * FROM t_part WHERE rowid = 'AAAAAAAABP//
AAAEEYAAAAAAAAAAO';
                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: c1, c2
   Node/s: datanode2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  ->  Partition Iterator
        Output: c1, c2
        Iterations: PART
        ->  Partitioned Index Scan using t_part_rowno_idx on public.t_part
              Output: c1, c2
              Distribute Key: c1
              Index Cond: (t_part.rowno = 14::bigint)
              Filter: ((t_part.rowid = 'AAAAAAAABP//AAAEEYAAAAAAAAAO'::rowid) AND (t_part.tableoid =
16664::oid))
              Selected Partitions:  1..4 (pbe-pruning)
(12 rows)


-- Clean up data.
ora=# DROP TABLE t_part;
DROP TABLE
ora=# RESET max_datanode_for_plan;
RESET
ora=# RESET query_dop;
RESET
ora=# RESET enable_light_proxy;
RESET
ora=# RESET enable_fast_query_shipping;
RESET
ora=# RESET enable_stream_operator;
RESET
ora=# RESET plan_cache_mode;
RESET
ora=# RESET auto_explain_level;
RESET
ora=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "$user".
gaussdb=# DROP DATABASE ora;
DROP DATABASE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8 Basic Concepts

8.1 Database Objects

A database object is a collection of logical structures of a database. Common database objects include tables, views, indexes, sequences, stored procedures, packages, and users.

Databases

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

Tablespaces

In GaussDB, a tablespace is a directory storing physical files of the databases. An instance may contain multiple tablespaces. Files are physically isolated using tablespaces and managed by a file system.

Tables

A table is a set of two-dimensional arrays. It is the basic unit of data organization and consists of rows and columns. A row of data in a database table is called a record, which consists of information of the same type.

- Tables are classified into ordinary tables and temporary tables.
  - The lifecycle of a temporary table exists only during a session. After the session is disconnected, the temporary table is automatically deleted.
  - The name of a temporary table can be the same as that of an ordinary table. If a temporary table and an ordinary table with the same name exist at the same time, operations such as viewing the table structure and DML operations are performed on the temporary table.
- Columns

  A column is used to record the value of an attribute in a table. The name defined for each attribute is called a column name. In addition to the column

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

name, the column also contains information such as the data type and the maximum length (precision) of the data type.

- Rows

  A row is a collection of all columns in a single record in a table. Each row in the table represents a group of related data and has a same structure.

Views

A view is a virtual table that does not store data. In essence, a view is a predefined query statement. You can retrieve data by accessing the view. A table involved in the SELECT statement is called a base table. If the data in the base table changes, the view changes accordingly. Deleting a view does not affect the base table.

- Advantages
  - Simplicity: Views simplify data understanding and operations. Common queries can be defined as views. You do not need to specify all conditions each time.
  - Security: Users can access only the visible data in the view, and other columns in the table are invisible to users. In this way, fine-grained permission control is implemented.
- Disadvantages
  - Performance issues: Do not use nested views in views because this may deteriorate query performance.
  - Maintenance complexity: Check whether the view is available after the base table definition is modified.

Indexes

An index is a data structure that contains reference pointers to all records in a data table. Similar to a book catalog, an index is used to accelerate database query and improve database performance.

- Advantages
  - The query speed is accelerated and the database I/O cost is reduced.
  - The sorting cost of the database and the grouping and sorting time are reduced.
  - A unique index ensures the uniqueness of data in a table.
- Disadvantages
  - Indexes require extra storage space.
  - It takes more time to insert and modify data.
- Index availability and visibility
  - Index availability

    In the scenario of deleting a partition, if the column whose index needs to be rebuilt is not specified, the index is marked as **UNUSABLE** (unavailable). In DML operations, indexes do not need to be maintained, and the optimizer ignores the indexes.
  - Index visibility

    The visibility of an index specifies whether the optimizer ignores the index. If an index is invisible, the optimizer ignores it. However, the index

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

still needs to be maintained in DML operations. Generally, before deleting an index, you can set the index to the invisible state to observe the impact on services. If there is no impact, delete the index.

Users and Roles

- A user is the access subject of a database. Each user has its own username and password. Users can perform database operations, and their permissions usually come from the roles granted to them.
  - Initial user

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

    NOTE

    - The OID of the initial user is 10, which can be queried from the gs_roles view.
    - An initial user bypasses all permission checks. It is recommended that this user be used only as a database administrator for database management instead of service applications.
  - SYSADMIN

    A system administrator is an account with the SYSADMIN attribute. By default, a system administrator has the same permissions as the object owner but does not have the object permissions in the dbe_perf schema.

    To create a security administrator when separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database as a security administrator and use a statement with the **CREATEROLE** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

    ```
    -- Specify the user as SYSADMIN during creation.
    gaussdb=# CREATE USER u1 WITH SYSADMIN PASSWORD '********';

    -- After creating a user, run the ALTER statement to specify the user as SYSADMIN.
    gaussdb=# CREATE USER u2 PASSWORD '********';
    gaussdb=# ALTER USER u2 SYSADMIN;

    -- Drop.
    gaussdb=# DROP USER u1;
    gaussdb=# DROP USER u2;
    ```
  - CREATEROLE

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

To create a security administrator when separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database as a security administrator and use a statement with the **CREATEROLE** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

```
-- Specify the user as CREATEROLE during creation.
gaussdb=# CREATE USER u1 WITH CREATEROLE PASSWORD '********';

-- After creating a user, run the ALTER statement to specify the user as CREATEROLE.
gaussdb=# CREATE USER u2 PASSWORD '********';
gaussdb=# ALTER USER u2 CREATEROLE;

-- Drop.
gaussdb=# DROP USER u1;
gaussdb=# DROP USER u2;
```

- AUDITADMIN

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

  If you want to create an audit administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database only as the initial user and use a statement with the **AUDITADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

  ```
  -- Specify the user as AUDITADMIN during creation.
  gaussdb=# CREATE USER auditadmin WITH AUDITADMIN password '********';

  -- After creating a user, run the ALTER statement to specify the user as AUDITADMIN.
  gaussdb=# CREATE USER u2 PASSWORD '********';
  gaussdb=# ALTER USER u2 AUDITADMIN;

  -- Drop.
  gaussdb=# DROP USER auditadmin;
  gaussdb=# DROP USER u2;
  ```

- MONADMIN

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

  To create a monitor administrator, connect to the database as a system administrator and use a statement with the **MONADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

  ```
  -- Specify the user as MONADMIN during creation.
  gaussdb=# CREATE USER u1 WITH MONADMIN password '********';

  -- After creating a user, run the ALTER statement to specify the user as MONADMIN.
  gaussdb=# CREATE USER u2 PASSWORD '********';
  gaussdb=# ALTER USER u2 MONADMIN;

  -- Drop.
  gaussdb=# DROP USER u1;
  gaussdb=# DROP USER u2;
  ```

- OPRADMIN

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

To create an O&M administrator, connect to the database as the initial user and use a statement with the **OPRADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

```
-- Specify the user as OPRADMIN during creation.
gaussdb=# CREATE USER u1 WITH OPRADMIN password '********';

-- After creating a user, run the ALTER statement to specify the user as OPRADMIN.
gaussdb=# CREATE USER u2 PASSWORD '********';
gaussdb=# ALTER USER u2 OPRADMIN;

-- Drop.
gaussdb=# DROP USER u1;
gaussdb=# DROP USER u2;
```

– POLADMIN

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

To create a security policy administrator, connect to the database as a system administrator and use a statement with the **POLADMIN** option. For details, see "SQL Reference > SQL Syntax > C > CREATE USER" or "SQL Reference > SQL Syntax > A > ALTER USER" in *Reference*.

```
-- Specify the user as POLADMIN during creation.
gaussdb=# CREATE USER u1 WITH POLADMIN password '********';

-- After creating a user, run the ALTER statement to specify the user as POLADMIN.
gaussdb=# CREATE USER u2 PASSWORD '********';
gaussdb=# ALTER USER u2 POLADMIN;

-- Drop.
gaussdb=# DROP USER u1;
gaussdb=# DROP USER u2;
```

– Common users: These users are usually granted the permissions required for performing specific service operations. You can run the CREATE USER statement to create a common user and specify the password, validity period, and connection restrictions.

```
-- Create user u1.
gaussdb=# CREATE USER u1 PASSWORD '********';

-- Drop.
gaussdb=# DROP USER u1;
```

- A role is a set of permissions used to organize and manage permissions. You can grant a role to a user so that the user has all the permissions of the role. Permissions of roles can be inherited.
  – Public role: GaussDB provides an implicitly defined role public. All created users and roles belong to the public role by default. You can run the GRANT and REVOKE statements to modify the PUBLIC permission.
  – Built-in roles: GaussDB provides a group of default roles whose names start with **gs_role_**. These roles provide access to specific high-privilege operations. For example, the gs_role_signal_backend role allows you to call functions to cancel or terminate other sessions, while the gs_role_tablespace role allows you to create tablespaces.
- Relationship between users and roles: The permissions of a user depend on the role of the user. A user can be regarded as a role with the login permission, and a role can be regarded as a user without the login permission. Roles facilitate permission management and grant, especially when multiple

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

users require the same permission. With these mechanisms, GaussDB implements flexible and secure permission management, enabling database administrators to properly allocate and manage user permissions based on service and security requirements.

```
-- Create role r1.
gaussdb=# CREATE ROLE r1 PASSWORD '********';

-- Create user u1.
gaussdb=# CREATE USER u1 PASSWORD '********';

-- Grant the permission of role r1 to user u1.
gaussdb=# GRANT r1 TO u1;

-- Query the information about the preceding role and user.
gaussdb=# \du r1|u1;
           List of roles
 Role name |  Attributes  | Member of
-----------+--------------+-----------
 r1        | Cannot login | {}
 u1        |              | {r1}

-- Drop the user and role.
gaussdb=# DROP USER u1;
gaussdb=# DROP ROLE r1;
```

8.2 Schema

Description

A schema is a database object. It is used to organize objects such as tables, views, indexes, functions, and operators in a database. The functions of a schema are as follows:

- Logical isolation: Database objects are organized into logical groups that are easy to manage. Different schemas can contain objects with the same name without conflicts.
- Permission control: Schemas can be used to control users' access to databases in a finer granularity. Permissions can be granted or revoked at the schema level to ensure data security and isolation.
- Easy management: The schema makes database object management more convenient and flexible, especially in the multi-application and multi-user environment.

Creating and Managing Schemas

The following describes only some common syntaxes. For details, see "CREATE SCHEMA", "ALTER SCHEMA", and "DROP SCHEMA" in "SQL Reference > SQL Syntax" in *Reference*.

- Create a schema.

```
gaussdb=# CREATE SCHEMA sch1 ;
CREATE SCHEMA

-- View the schema information.
gaussdb=# \dn sch1
List of schemas
 Name | Owner
------+-------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 sch1 | omm
(1 row)
```

- Modify a schema.

```
-- Perform the pre-operation to create a user.
gaussdb=# CREATE USER u1 PASSWORD '********';
CREATE ROLE

-- Rename a schema.
gaussdb=# ALTER SCHEMA sch1 RENAME TO sch2;
ALTER SCHEMA

-- Change the owner of a schema.
gaussdb=# ALTER SCHEMA sch2 OWNER TO u1;
ALTER SCHEMA

-- View the schema information.
gaussdb=# \dn sch*
List of schemas
 Name | Owner
------+-------
 sch2 | u1
(1 row)
```

- Drop the schema.

```
gaussdb=# DROP SCHEMA sch2;
DROP SCHEMA

-- Drop the user.
gaussdb=# DROP USER u1;
DROP ROLE
```

Search Path

The search path is used to determine the sequence in which the database searches for database objects in the corresponding schema when parsing database object names. The setting of the search path can affect the analysis and execution of the SQL query. When a user uses an object (such as a table, view, index, or function) without specifying a schema, the database searches for the object in the sequence defined in the search path until the object is found.

- Query the search path.

  In the following example of the query result, if no schema name is specified, the database searches for objects in the schema with the same name as the username and then in the public schema.

```
-- Query the current search path.
gaussdb=# SHOW search_path;
  search_path
----------------
 "$user",public
(1 row)
```

- Set the search path.
  - The search path setting takes effect only in the current session and does not affect other sessions.
  - If no schema is specified when an object is created, the object is created in the first schema in the search path by default.

```
-- Create schemas.
gaussdb=# CREATE SCHEMA sch1;
CREATE SCHEMA
gaussdb=# CREATE SCHEMA sch2;
CREATE SCHEMA

-- Create tables with the same name in the two schemas and insert data into the tables.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE TABLE sch1.tb_test(c1 int);
CREATE TABLE
gaussdb=# INSERT INTO sch1.tb_test VALUES (1);
INSERT 0 1
gaussdb=# CREATE TABLE sch2.tb_test(c1 int);
CREATE TABLE
gaussdb=# INSERT INTO sch2.tb_test VALUES (2);
INSERT 0 1

-- Set the sequence of the search path sch1 to the front, and then view the table data.
gaussdb=# SET search_path = sch1,sch2,public;
SET
gaussdb=# SELECT * FROM tb_test;
 c1
----
  1
(1 row)


-- Set the sequence of the multi-path sch2 to the front, and then view the table data.
gaussdb=# SET search_path = sch2,sch1,public;
SET
gaussdb=# SELECT * FROM tb_test;
 c1
----
  2
(1 row)

-- Reset search_path and drop schemas and tables.
gaussdb=# RESET search_path;
RESET
gaussdb=# DROP TABLE sch1.tb_test;
DROP TABLE
gaussdb=# DROP TABLE sch2.tb_test;
DROP TABLE
gaussdb=# DROP SCHEMA sch1;
DROP SCHEMA
gaussdb=# DROP SCHEMA sch2;
DROP SCHEMA
```

8.3 Constraints

SQL constraints are used to specify the data rules in a table. If a user attempts to insert data that violates a constraint into a table, an error is thrown. Constraints are classified into the following types:

- NOT NULL constraints
- UNIQUE constraints
- PRIMARY KEY constraints
- CHECK constraints

NOT NULL Constraints

- Columns with NOT NULL constraints cannot contain NULL values.
- Only column-level NOT NULL constraints can be created.
- Multiple NOT NULL constraints can be defined in the same table.

```
-- Create a table and add a NOT NULL constraint.
gaussdb=# CREATE TABLE tbl_person1(
    id int not null,
    name varchar(50)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

-- Insert data that does not comply with the constraint. An error is reported.
gaussdb=# INSERT INTO tbl_person1 (id,name) VALUES (NULL,'Bob');
ERROR:  dn_6001_6002_6003: The null value in column "id" violates the not-null constraint.
DETAIL:  Failing row contains (null, Bob).

-- Drop the table.
gaussdb=# DROP TABLE tbl_person1;
DROP TABLE
```

UNIQUE Constraints

- Columns with UNIQUE constraints cannot contain duplicate values.
- Multiple UNIQUE constraints can be defined in the same table.
- After a UNIQUE constraint is defined, the database automatically adds a UNIQUE index to the constraint column.
- A column can have both UNIQUE and NOT NULL constraints. If a column has only the UNIQUE constraint, the column can contain multiple **NULL** values.
- Multiple UNIQUE constraints can be defined in a table.

```
-- Create a table and add a column-level UNIQUE constraint.
gaussdb=# CREATE TABLE tbl_person2(
    id int UNIQUE,
    name varchar(50)
);
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "tbl_person2_id_key" for table "tbl_person2"
CREATE TABLE

-- Create a table and add a table-level UNIQUE constraint.
gaussdb=# CREATE TABLE tbl_person3(
    id int,
    name varchar(50),
    CONSTRAINT uq_person3_id UNIQUE(id)
);
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "uq_person3_id" for table "tbl_person3"
CREATE TABLE

-- Insert duplicate values. A UNIQUE constraint error is reported.
gaussdb=# INSERT INTO tbl_person2 VALUES (1,'Bob');
INSERT 0 1
gaussdb=# INSERT INTO tbl_person2 VALUES (1,'Bob');
ERROR:  dn_6004_6005_6006: Duplicate key value violates unique constraint "tbl_person2_id_key".
DETAIL:  Key (id)=(1) already exists.

-- Duplicate NULL values do not trigger a UNIQUE constraint.
gaussdb=# INSERT INTO tbl_person2 VALUES (null,'Bob');
INSERT 0 1
gaussdb=# INSERT INTO tbl_person2 VALUES (null,'Bob');
INSERT 0 1

-- Drop.
gaussdb=# DROP TABLE tbl_person2,tbl_person3;
DROP TABLE
```

PRIMARY KEY Constraints

- Columns with PRIMARY KEY constraints cannot contain duplicate values and **NULL** values.
- You can define PRIMARY KEY constraints for one or more columns.
- After a PRIMARY KEY constraint is created, the database automatically creates a UNIQUE index and a NOT NULL constraint for the column.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Only one PRIMARY KEY constraint can be defined in a table.

```
-- Create a table and add a column-level PRIMARY KEY constraint.
gaussdb=# CREATE TABLE tbl_person4(
    id int PRIMARY KEY,
    name varchar(50)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tbl_person4_pkey" for table "tbl_person4"
CREATE TABLE

-- Create a table and add a table-level PRIMARY KEY constraint.
gaussdb=# CREATE TABLE tbl_person5(
    id int,
    name varchar(50),
    CONSTRAINT pk_person5_id PRIMARY KEY(id)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "pk_person5_id" for table "tbl_person5"
CREATE TABLE

-- Drop the table.
gaussdb=# DROP TABLE tbl_person4,tbl_person5;
DROP TABLE
```

CHECK Constraints

- An expression with a CHECK constraint must reference one or more columns in a table, and the calculation result of the expression must be a Boolean value. The expression cannot contain subqueries.
- CHECK constraints can be defined at the column or table level.
- Both CHECK and NOT NULL constraints can be defined for the same column.
- Multiple CHECK constraints can be defined for the same column.
- Multiple CHECK constraints can be defined in the same table.

```
-- Create a table and add a CHECK constraint.
gaussdb=# CREATE TABLE tbl_person6(
    id int PRIMARY KEY,
    name varchar(50) NOT NULL,
    age int CHECK(age > 0 AND age < 200) NOT NULL
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tbl_person6_pkey" for table "tbl_person6"
CREATE TABLE

-- Insert data that does not comply with the constraint. An error is reported.
gaussdb=# INSERT INTO tbl_person6 VALUES (1,'Bob',500);
ERROR:  dn_6004_6005_6006: New row in relation "tbl_person6" violates check constraint
"tbl_person6_age_check".
DETAIL:  N/A

-- Drop the table.
gaussdb=# DROP TABLE tbl_person6;
DROP TABLE
```

8.4 Sequences

Description

A sequence is a database object used to generate a numeric queue. A value of the sequence may increase or decrease based on a specified increment. By default, the maximum value of **Sequence** is 2^{63} – 1. If a large identifier is used, the maximum value can be 2^{127} – 1.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

- Creating and using a sequence

```
-- Create a sequence.
gaussdb=# CREATE SEQUENCE seq_test
    START WITH 1
    INCREMENT BY 1
    MAXVALUE 99999;

-- View the next value of the sequence.
gaussdb=# SELECT nextval('seq_test');
 nextval
---------
       1
(1 row)

-- Drop the sequence.
gaussdb=# DROP SEQUENCE seq_test;
DROP SEQUENCE
```

- Sequence gaps

  After the sequence cache is set, the performance can be improved, but gaps may also occur.

```
-- Create a table.
gaussdb=# CREATE TABLE tb_test(c1 int,c2 varchar);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Create a sequence joined with the table.
gaussdb=# CREATE SEQUENCE seq_test
    START 1
    CACHE 10
    OWNED BY tb_test.c1;
CREATE SEQUENCE

-- Set the default value of a column.
gaussdb=# ALTER TABLE tb_test ALTER COLUMN c1 SET DEFAULT nextval('seq_test'::regclass);
ALTER TABLE

-- Insert data.
gaussdb=# INSERT INTO tb_test (c2) VALUES ('a');
INSERT 0 1
gaussdb=# INSERT INTO tb_test (c2) VALUES ('b');
INSERT 0 1

-- Log out of the database, log in again, and insert data again.
gaussdb=# INSERT INTO tb_test (c2) VALUES ('c');
INSERT 0 1

-- Query data. Gaps exist in the sequence.
gaussdb=# SELECT * FROM tb_test ORDER BY c1;
 c1 | c2
----+----
  1 | a
  2 | b
 11 | c
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE tb_test;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.5 System Columns

System columns are automatically created and maintained by the database system in database tables. These columns are used to store metadata related to tables. The system columns are as follows:

- **oid**: object identifier, which is the unique identifier of each table in the database.
- **ctid**: row identifier, which is used to identify a row in a table. Each row has a unique **ctid**.
- **xmin**: transaction identifier, indicating the minimum transaction ID of the row inserted into the table.
- **xmax**: transaction identifier, indicating the maximum transaction ID for deleting or updating a row in a table.
- **cmin** and **cmax**: command identifiers, indicating the IDs of the commands for inserting and deleting a row, respectively.
- **xc_node_id**: node identifier, which is used to identify the physical node where the data is located.
- **rowid**: globally unique logical row identifier, which is dynamically computed and generated based on **slice_id**, **bucket_id**, **table_oid**, and **rowno**.
- **rowno**: unique physical row number in a table, which is automatically generated by a dedicated sequence and persistently stored.

8.6 Partitioned Tables

Description

Database table partitioning is a database optimization technology. It can physically divide a large table into multiple smaller parts to improve query performance. Each part is called a partition, and these partitions can be stored on different storage devices.

- **Advantages:**
  - Improving query performance: The query performance is significantly improved by reducing the amount of data to be scanned.
  - Optimizing storage: Different partitions are stored on different storage media to balance performance and costs.
  - Improving maintainability: Partitioned table maintenance operations (data cleanup and index rebuilding) can be performed based on the partition granularity to reduce the impact on the entire system.
  - Improving concurrency: Partitioned tables can improve concurrency because multiple partitions can be processed in parallel. For example, multiple queries can access different partitions at the same time without interfering with each other.
- **Disadvantages:**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Memory usage: The memory used by a partitioned table is about (Number of partitions x 3/1024) MB. If the memory is insufficient due to too many partitions, the performance deteriorates sharply.
  - Partitioning policy complexity: Technical knowledge and experience are required to develop and implement appropriate partitioning policies. If an improper partitioning policy is selected, data may be unevenly distributed, which further affects performance.
  - Complexity of backup and restoration: Although partitions can be backed up and restored separately, more detailed backup policies and management are required.

Precautions

- **Application scenarios where partitioned tables are used**
  - Improving query performance: If a table (for example, a table that is analyzed by month, quarter, or year) contains a large amount of data and data with certain features is frequently used in a certain scenario, you can reduce the amount of data to be scanned during query to improve performance.
  - Balancing between performance and cost: The data volume of a table is too large. Cold data (infrequently accessed data) needs to be moved to low-cost storage, and hot data (frequently accessed data) needs to be retained on high-performance storage.
  - Managing tables with a large amount of data: A large amount of data in a table needs to be stored on multiple storage media.
- **Partitioning key selection:**

  Selecting the partition key of a partitioned table is an important design decision because it directly affects the performance, maintainability, and data management efficiency of the database.
  - Query optimization: Select the partition key for common queries. For example, you can select the date column as the partition key for a table that is frequently queried by date.
  - Data distribution: When selecting a partition key, consider the data distribution. This prevents the situation where the data volume of some partitions is too large and that of other partitions is too small.
  - Partition quantity and management: Control the number of partitions to prevent too many partitions from being created. Too many partitions will increase management complexity and deteriorate performance.

Partitioned Table Types

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

- Range partitioned table: Data in different ranges is mapped to different partitions. The range is determined by the partition key specified during the partitioned table creation. It is often used for time series data, such as partitioning by date, month, or year.
- List partitioned table: Data is allocated to different partitions based on the partition key values. The key values contained in a partition are specified

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

when the partitioned table is created. This applies to well-classified data, such as order status, device type, or region code.

- Hash partitioned table: Data is mapped to different partitions based on the hash algorithm. The number of partitions is specified when the partitioned table is created. It applies to scenarios where data needs to be evenly distributed to balance loads.

Examples

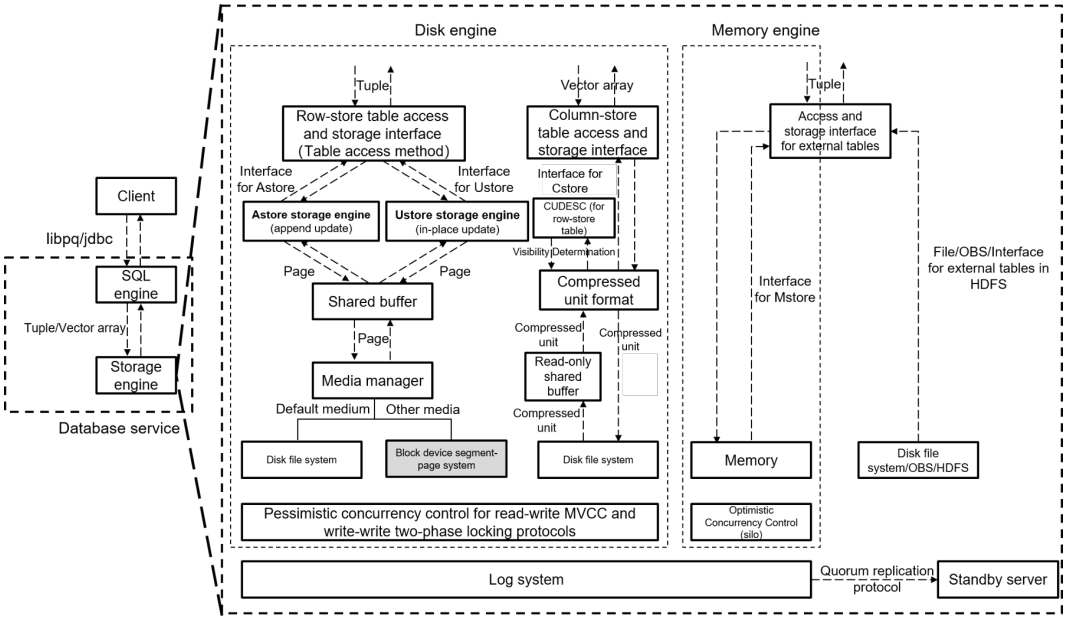
```
-- Example 1 of a range partitioned table.
gaussdb=# CREATE TABLE test_range1(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200),
    PARTITION p3 VALUES LESS THAN (300),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Example 2 of a range partitioned table.
gaussdb=# CREATE TABLE test_range2(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 START(1) END(600) EVERY(200),
    PARTITION p2 START(600) END(800),
    PARTITION pmax START(800) END(MAXVALUE)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Example of a list partitioned table.
gaussdb=# CREATE TABLE test_list ( NAME VARCHAR ( 50 ), area VARCHAR ( 50 ) )
PARTITION BY LIST (area) (
    PARTITION p1 VALUES ('Beijing'),
    PARTITION p2 VALUES ('Shanghai'),
    PARTITION p3 VALUES ('Guangzhou'),
    PARTITION p4 VALUES ('Shenzhen'),
    PARTITION pdefault VALUES (DEFAULT)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'name' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Example 1 of a hash partitioned table.
gaussdb=# CREATE TABLE test_hash1(c1 int) PARTITION BY HASH(c1) PARTITIONS 3;
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Example 2 of a hash partitioned table.
gaussdb=# CREATE TABLE test_hash2(c1 int) PARTITION BY HASH(C1)(
    PARTITION pa,
    PARTITION pb,
    PARTITION pc
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Drop tables.
gaussdb=# DROP TABLE test_range1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
DROP TABLE
gaussdb=# DROP TABLE test_range2;
DROP TABLE
gaussdb=# DROP TABLE test_list;
DROP TABLE
gaussdb=# DROP TABLE test_hash1;
DROP TABLE
gaussdb=# DROP TABLE test_hash2;
DROP TABLE
```

8.7 Storage Engine

The database storage engine is responsible for data storage, retrieval, and management. From the perspective of the entire database service architecture, the storage engine interfaces upward with the SQL engine to exchange data in a standard format (tuple or vector array), and downward reads or writes data in units of pages, compression units (CUs), or other forms through APIs provided by the storage media. GaussDB enables database professionals to select dedicated storage engines for meeting specific application requirements through static compilation. To reduce interference to the execution engines, the row-store table access method (TableAM) layer is provided to shield the differences caused by the underlying row-store engines so that different row-store engines can evolve independently. See the following figure.



On this basis, the storage engines provide data persistence and reliability assurance through the log system. The concurrency control (transaction) system ensures atomicity, consistency, and isolation between concurrent read and write operations. The index system provides accelerated addressing and query capabilities for specific data. The primary/standby replication system provides high availability of the entire database service.

Row-store engines are oriented to online transaction processing (OLTP) scenarios, which are suitable for highly concurrent read and write operations on a small amount of data at a single point or within a small range. Row-store engines provide APIs upward to read tuples from or write tuples to the SQL engine, perform read and write operations downward on storage media by page through

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

an extensible media manager, and improve read and write operation efficiency in the shared buffer by page. For concurrent read and write operations, multi-version concurrency control (MVCC) is used. For concurrent write and write operations, pessimistic concurrency control (PCC) based on the two-phase locking (2PL) protocol is used.

Currently, the default media manager of row-store engines uses the disk file system API. Other types of storage media such as block devices will be supported in the future. The GaussDB row-store engine can be the append update-based Astore or in-place update-based Ustore.

Why Is a Storage Engine Needed?

Different services have different requirements for data writing and updating. Some are write-intensive with fewer updates, focusing on write throughput and the simplicity of exception recovery. Others involve high-frequency updates, focusing on space bloat control and stable transaction latency. A single data organization method can hardly balance both write efficiency and space bloat control during updates, making it difficult to meet these conflicting requirements.

If a database features only one fixed built-in storage method, it can only perform well in specific scenarios, making it difficult to adapt to diversified services. To address this, GaussDB adopts a pluggable storage engine architecture. Table Access Method APIs are abstracted on top of unified transaction, log, concurrency control, and cache management to decouple data storage from transaction management. Users can select either the Astore or Ustore on a per-table basis. In this way, tables with different workload characteristics within the same database instance can use their most suitable storage engine, balancing performance, space, and reliability.

Storage Engine Classification

In GaussDB, storage engines are classified into Ustore and Astore. Ustore is an in-place update engine and serves as the default storage engine of GaussDB Kernel. It stores old and new versions separately, controls space bloat, supports full flashback, and is ideal for high-concurrency update scenarios. Conversely, Astore is an append update engine, featuring efficient writes and simple restoration.

Ustore

Ustore uses the in-place update mode, which is a new storage mode added to the storage kernel. Different from the append update mode, Ustore performs well in scenarios with frequent UPDATE (in-page updates).

The main idea behind in-place update storage mode is to store the valid data of the latest version and the junk data of historical versions separately. The valid data of the latest version is stored on the data pages, and a separate undo space is created for managing the junk data of historical versions. Therefore, the data space does not bloat due to frequent updates, and the junk data is recycled more efficiently. The NUMA-aware undo subsystem design enables the undo subsystem to efficiently scale on multi-core platforms. In addition, the tuple and data page structures are redesigned to reduce storage space occupation. The index versioning technology is used to solve the index bloat problem, eliminate the dependency on the autovacuum (asynchronous cleaner thread) mechanism, and improve the storage space recycling and reuse efficiency.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Advantages of Ustore

- **Controllable space bloat**: The latest and historical versions are stored separately. Historical versions are centrally managed by the undo space and can be reclaimed in batches. This effectively suppresses the space bloat of data pages and indexes under frequent updates.
- **In-place update and stable CTID**: Heap table updates for both non-indexed and indexed columns can be completed in place, keeping the tuple CTID relatively stable. When the same row is updated under high concurrency, a first-come, first-served mechanism applies, ensuring relatively stable update latency.
- **Smaller scan scope and fewer table lookups**: The multi-version B-tree index (carrying transaction information and supporting independent MVCC) helps increase the proportion of index-only scans, drastically reducing table lookups.
- **Autonomous space reclamation**: Old versions are cleared without relying on VACUUM. Indexes are decoupled from heap tables and can be cleaned up independently, ensuring smoother I/O.
- **Enterprise-level flashback**: Combined with the undo space, it enables more efficient and comprehensive flashback queries and flashback tables, supporting fast rollback of misoperations.

Application Scenarios of Ustore

- **OLTP scenarios with frequent updates**: In write/delete-intensive services such as order and accounting, the combination of in-place updates and separate undo storage effectively suppresses data page and index bloat.
- **High-concurrency updates to the same row**: The in-place update mechanism ensures a stable tuple CTID and operates on a first-come, first-served basis, maintaining relatively stable update latency.
- **Requirements for complete flashback and recycle bin capabilities**: Relying on the undo space, it delivers more efficient and comprehensive enterprise-level recovery functions, such as flashback queries and flashback tables.
- **High requirements for space controllability**: The dependency on VACUUM is eliminated through dynamic page cleanup during DML operations. Autonomous space management prevents I/O jitter caused by asynchronous cleanup.

Working Principles of Ustore

The following table describes the main functional modules.

Module	Description
Ustore table access management	Connects upward to the SQL engine to provide APIs for row-level query, insertion, deletion and modification of Ustore tables. Downward, it implements the traversal, insertion, deletion, modification, and query of specified Ustore table files based on the inter-page and intra-page structures as well as tuple structure of index tables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Ustore index access management	Connects upward to the SQL engine to provide APIs for row-level query, insertion, and deletion of index tables. Downward, it implements the query, insertion, and deletion of specified index keys based on the inter-page and intra-page structures as well as tuple structure of index tables.
Ustore table page structure	Manages the organization of Ustore table tuples on the page, tuple insertion on the page, page arrangement, and page initialization.
Ustore table tuple structure	Manages the structure, filling, deconstruction, modification, column query, and deformation of Ustore table tuples.
Undo management	Manages the structure, filling, encoding, and decoding of undo records.
Multi-version indexing	Manages the page structure, query, modification, visibility check, and space recycling for Ustore-dedicated multi-version UB-tree indexing.

Example

- Create a Ustore table, insert some data, and view the tuple location information.

```
-- Create a Ustore table.
 gaussdb=# CREATE TABLE tb_t1(id int, name VARCHAR(20)) WITH (STORAGE_TYPE=Ustore);
 CREATE TABLE

-- Insert data.
 gaussdb=# INSERT INTO tb_t1 VALUES (1,'Joe'),(2,'Jack');
 INSERT 0 2

-- Check the ctid information of a tuple.
-- ctid(blkno,offset) indicates the location of the tuple on the data page. blkno indicates the page number, and offset indicates the offset of the tuple on the page.
gaussdb=# select *, ctid from tb_t1;
 id | name | ctid
----+------+-------
  1 | Joe  | (0,1)
  2 | Jack | (0,2)
(2 rows)
```

- Modify a record and check the tuple location information. You can see that the tuple **ctid** does not change, that is, the tuple is updated in-place. The old data before the update is stored in the undo space.

```
-- Modify the tuple whose ID is 2.
gaussdb=# UPDATE tb_t1 SET name='Scott' WHERE id=2;
UPDATE 1

-- Check the ctid information again.
gaussdb=# select *, ctid from tb_t1;
  id | name  | ctid
----+-------+-------
  1 | Joe   | (0,1)
  2 | Scott | (0,2)
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Rollback Segment and MVCC

- **Rollback segment**

  Old data is stored in the **undo** directory of the rollback segment. To reduce read/write conflicts, earlier version data (rollback records) is appended to the **undo** directory under the data directory. In this way, the read and write of the old data do not conflict, and the old data of the same transaction is stored continuously, facilitating the rollback operation. To reduce contention during concurrent write, the **undo** directory space is divided into multiple undo zones (rollback segment logical zones). Threads are allocated to their own logical zones and are completely isolated from other threads. In this way, no extra lock overhead is required when old data is written to the allocated space. The undo zones may be further divided based on the NUMA cores of the CPU, and each thread is allocated from the undo zone on the current NUMA core, further improving allocation efficiency. When the undo space is allocated, the action is recorded at the transaction granularity. Once the old data is confirmed not to be accessed by any transaction, the space is recycled.

- **MVCC**

  The visibility check of Ustore is similar to that of Astore. The snapshot CSN is compared with the CSN of the tuple deletion and insertion transactions to determine whether the tuple is visible. Ustore and Astore use the same transaction management mechanism and snapshot management mechanism.

  The biggest difference between Ustore and Astore is that Astore stores earlier version data on pages, while Ustore stores earlier version data in the rollback segment. When earlier version data needs to be obtained, Astore can read the transaction IDs (XIDs) of tuple insertion and deletion from the tuple header to determine the visibility of the tuple. However, Ustore needs to read the transaction information of the earlier version from the rollback segment to determine whether the earlier version is visible. Because the cost of reading earlier version data from the rollback segment is relatively high, Ustore uses a series of optimization methods to avoid reading earlier version data from the rollback segment.

  When Ustore obtains a tuple, it checks the corresponding transaction directory. Transaction directories can be valid or invalid. If the transaction directory is valid, Ustore obtains the latest transaction on the tuple.

  If the transaction directory is frozen, the tuple is visible in all transactions. If the transaction ID in the transaction directory is smaller than **oldestXidInUndo**, the tuple is old enough to be visible in all transactions. In addition, the transaction directory is frozen to accelerate subsequent queries.

  If a tuple is marked with an invalid transaction directory, the transaction used to modify the tuple has been committed and is older than the transaction in the current transaction directory. In this case, Ustore uses the transaction in the transaction directory to determine the visibility. If the transaction is visible, the transaction used to modify the tuple is visible. In this case, you do not need to read the transaction information from the **undo** directory.

Space Management and Reclamation

Unlike the space management and recycling mechanism of Astore, Ustore implements an autonomous space management mechanism. In Ustore, heap and index space allocation and reclamation are smoothly performed during service runtime, without relying on the VACUUM or AUTOVACUUM mechanism.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Autonomous heap page space management**

  Autonomous heap page space management in Ustore is based on a lightweight heap page cleanup mechanism that closely resembles the one used in Astore. During DML and DQL operations, Ustore clears heap pages, replacing the VACUUM mechanism. Ustore clears committed tuples that have been deleted.

  For Astore, the corresponding index tuple must be cleared before the row pointer of a tuple is reused. This is to prevent incorrect data from being retrieved by accessing the reused row pointer through the index tuple. In Astore, the VACUUM operation is required to clear invalid index tuples so that the row pointer can be reused. This couples the cleanup logic of heap pages and index pages, causing intermittent heavy I/O. In Ustore, data and index pages can be cleared separately. The UB-tree with version information can independently detect and filter out invalid index tuples, and does not access the corresponding data table through invalid index tuples.

  The space management mechanism of heap pages reuses the FSM in GaussDB to manage the available space in U-Heap. After the pages are cleared, the free space is released to the corresponding FSM pages. To avoid extra overhead caused by updating the entire FSM in the tree structure each time a page is cleared, a probability calculation for updating the entire FSM is introduced. The probability of updating the entire FSM after a page is cleared is calculated based on the percentage of the available space after the current cleanup to the reserved free space threshold. In other words, a larger available space obtained through page cleanup indicates a higher probability of updating the entire FSM.

  When a data tuple is deleted, the potential free space on the page is recorded, which is used to estimate the free space on the page. During runtime, the system attempts to clear the page in multiple scenarios. During the execution of DML statements, INSERT, UPDATE, and DELETE operations obtain the write lock of the page. If the space is insufficient or the potential free space reaches a threshold, the system attempts to clear the page. During the execution of a DQL query statement, the system also attempts to apply for a write lock for the page if the potential free space on the page reaches the threshold. After obtaining the write lock, the system attempts to clear the page.

  The current clearance mechanism is based on access. Some pages have tuples that can be cleared, but are not accessed. As a result, the tuples cannot be properly cleared using this mechanism. To resolve this problem, a probability-based cleanup solution is introduced. In some operations of searching for new available pages, if FSM finds that the free space is insufficient, it will randomly select some pages for cleanup before the physical file is expanded. The cleanup algorithm can assist in finding these potential pages that need to be cleaned up.

- **Autonomous index page space management**

  The index page space management does not depend on the FSM data structure, but depends on the unique UB-tree recycle queue (URQ) structure, which is called the recycle queue for short. The recycle queue in the index is stored in the .urq file corresponding to the UB-tree index. The .fsm file of the original B-tree index does not exist.

  The recycle queue in the index is divided into two parts: potential empty page queue and available page queue. Both queues are cross-page cyclic queues,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

where each element stores **blkno** and **XID**. **blkno** indicates the block number of the index page corresponding to an element. **XID** indicates the time when the page can be recycled or reused. The cyclic queue sorts these elements on each page based on **XID** so that a page with the smallest **XID** (most likely to be recycled or reused) can be quickly found.

The potential empty page queue stores pages whose tuples have been deleted but are not all invalid. **XID** indicates the possible time when the last tuple on the page becomes invalid. When the **oldestXmin** of the entire system exceeds **XID**, the page may be deleted from the index. The deletion may fail if the transaction used to insert or delete the tuple is terminated. After a page in the potential empty page queue is successfully deleted, the page is added to the available page queue, and the XID of the latest transaction at the time of the deletion is recorded.

The available page queue stores pages that have been deleted but can or will be reused. **XID** indicates the time when the page can be reused. The page reuse delay is caused by possible concurrent access when the UB-tree index page is deleted.

During service runtime, indexes continuously attempt to recycle pages in the potential empty page queue. When an index applies for a new page, it searches for available idle pages. If no available page is found in the available page queue, it usually obtains a new page by expanding a physical index file. However, physical files may be extended in batches, or the system exits due to an error before the extended files are used. At this time, the metadata page of the recycle queue records the number of pages that have been correctly tracked. If this number is less than the number of pages in the entire index table, the system attempts to use the untracked pages and update the number of tracked pages.

Astore

Append Store (Astore) is a storage engine design model for GaussDB that stores data in append mode. This means that data updates will be appended without overwriting the existing data.

Advantages of Astore

- **Efficient writes**: Adopting an append-only storage method, data is directly appended to the end, avoiding data movement and reconstruction. This ensures high write performance, making it ideal for scenarios with frequent inserts and few updates.
- **Simple exception recovery**: Without rollback segments, old data is kept in the original data files. If a database crash occurs, the recovery process is simpler because it does not require the synchronization of both redo and undo logs as Ustore does.
- **Fast rollback**: Since a rollback does not delete data, it can be completed quickly.
- **Simpler WALs**: Only changes to data files need to be recorded, while changes to rollback segments are not required.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Application Scenarios of Astore

- **Workloads with frequent writes and few updates**: In INSERT-dominant scenarios with few UPDATE/DELETE operations, such as logs, transaction history, and archiving, append-only writes can deliver high write throughput.
- **Historical data retention**: Data of older versions is not overwritten immediately, making it suitable for scenarios where historical records must be retained for easy data restoration and reconstruction.
- **High requirements for recovery simplicity**: Scenarios where the exception recovery and rollback processes need to be as simple and fast as possible. Note that Astore is prone to space bloat under frequent updates, requiring periodic cleanup using VACUUM.

Working Principles of Astore

Overall framework

As a sub-format of row store, Astore has its own heap table tuple structure, heap table page structure, multi-version tuple mechanism, and free space management and recycling mechanism.

Astore stores data in a heap table, without a specific sequence. Tuple updates are implemented in append mode on the same page, allowing both new and original tuples to coexist on Astore's heap table page.

The following table describes the main functional modules.

Module	Description
Astore access and storage management	Provides access and storage management for Astore row-store tables, including APIs for row-level query, insertion, deletion, and modification of Astore heap tables, row-level multi-version mechanism and tuple visibility determination for Astore heap tables, and traversal, addition, deletion, modification, and query of Astore heap table files based on the inter-page and intra-page structures as well as tuple structure of Astore heap tables.
Astore index access and storage management	Connects upward to the SQL engine to provide APIs for row-level query, insertion, and deletion of index tables. Downward, it implements the query, insertion, and deletion of specified index keys based on the inter-page and intra-page structures as well as tuple structure of index tables.
Astore heap table/ index table page structure	Manages the specific organization of Astore heap table/ index table tuples on the page, tuple insertion on the page, page arrangement, page initialization, page encryption and decryption, and page cyclic redundancy check (CRC).
Astore heap table tuple structure	Manages the structure, filling, deconstruction, modification, column query, deformation, compression, and decompression of Astore heap table tuples.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Space management	Works with Free Space Map (FSM) to support space management and control mechanisms such as lightweight online space management and asynchronous cleanup (Autovacuum).

Example

- Create an Astore table, insert some data, and view the tuple location information.

```
-- Create an Astore table.
gaussdb=# CREATE TABLE t1(id int, name VARCHAR(20)) WITH (STORAGE_TYPE=Astore);
CREATE TABLE
-- Insert data.
gaussdb=# INSERT INTO t1 VALUES (1,'Joe'),(2,'Jack');
INSERT 0 2
-- Check the ctid information of a tuple.
-- ctid(blkno,offset) indicates the location of the tuple on the data page. blkno indicates the page number, and offset indicates the offset of the tuple on the page.
gaussdb=# select *, ctid from t1;
 id | name | ctid
----+------+-------
  1 | Joe  | (0,1)
  2 | Jack | (0,2)
(2 rows)
```

- Modify a record and view the tuple location information.

```
-- Modify the tuple whose ID is 2.
gaussdb=# UPDATE t1 SET name='Scott' WHERE id=2;
UPDATE 1
-- Check the ctid information again.
gaussdb=# select *, ctid from t1;
 id | name  | ctid
----+-------+-------
  1 | Joe   | (0,1)
  2 | Scott | (0,3)
(2 rows)
```

The **ctid** of the updated tuple changes from **(0,2)** to **(0,3)**. A new tuple is appended to the end of the existing data without overwriting the existing data, which is known as non-in-place update. The existing data is still stored on the current page.

MVCC

Astore supports the multi-version concurrency control mechanism for tuples. That is, physical tuples of multiple historical versions are reserved for the same record to resolve concurrent read and write conflicts of the same record. (Read and write transactions work on physical tuples of different versions.)

Astore adopts an optimized append-write design, allowing both historical and current versions of tuples to coexist on the same page. Supposing a tuple is updated from v0 to v1, if the page of tuple v0 still has sufficient space, the updated tuple v1 will be inserted into that page, and the pointer of tuple v0 points to the pointer of tuple v1. Throughout this process, new tuple versions are intermixed with the updated older versions via append-only write operations, which minimizes the I/O overhead of update operations. However, it should be noted that because the new and old tuple versions are mixed, the cleanup overhead required when reclaiming old tuple versions is relatively high. Therefore, Astore is suitable for service scenarios with frequent insertions and a small number of updates.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Transaction information (such as **xmin** and **xmax**) is stored in Astore tuples. **xmin** indicates the transaction information when a tuple is inserted, and **xmax** indicates the transaction information when a tuple is deleted or updated. At the default isolation level (read committed), to determine whether a tuple is visible, you only need to determine whether the transaction information on the tuple is visible based on the commit sequence number (CSN). For example, if a tuple is updated from v0 to v1, and the tuple is queried by a snapshot in a concurrent transaction, the conditions for the tuple v0 to be visible are: the CSN corresponding to the **xmin** column is less than or equal to the CSN of the queried snapshot, and the CSN corresponding to the **xmax** column is greater than the CSN of the queried snapshot. Tuple v1 is visible as long as the CSN corresponding to the **xmin** column is less than or equal to the CSN of the queried snapshot (the **xmax** column of tuple v1 is **0** and therefore does not need to be considered). When the v0 tuple is invisible, the v1 tuple may be found by using the **ctid** recorded in the v0 tuple to determine whether the v1 tuple is visible.

Space management and recycling

Astore uses the maximum heap binary tree structure to record and manage the free space of heap table pages. The maximum heap binary tree structure performs read and write operations on the storage medium at the page granularity and is stored in a dedicated FSM.

All pages are classified into leaf node pages and internal node pages. The internal structures of the two types of pages are the same. The difference is as follows: For a leaf node page, the free space degree of the heap or index table page corresponding to the leaf node in the binary tree is recorded on the page. For an internal node page, the maximum free space degree of the lower-layer FSM page corresponding to the leaf node in the binary tree is recorded on the page.

One byte (that is, 256 levels) on the FSM page is used to record the free space degree of a heap or index page. The FSM page does not record the page number of any heap or index page, nor does it record the page number of any root or leaf FSM node page. This information is calculated according to the following rules:

1. On an FSM page, binary tree nodes are arranged layer by layer from top to bottom and from left to right. That is, the first byte indicates the free space degree of the root node, the second byte indicates the free space degree of the leftmost node at the first layer, and so on.

2. All FSM pages are stored in the physical storage in depth-first order. That is, all physical pages before an FSM page include all upper-layer nodes of the subtree where the FSM page is located and all left subtrees of the FSM page.

3. The leaf nodes in the binary trees on all leaf FSM node pages correspond to the free space degree of the heap or index table pages, and correspond to the first, second, ..., and *n*th heap or index table physical pages in left-to-right order.

4. Except for the FSM nodes described in **3**, other parent FSM nodes store the maximum free space degree of the child nodes (subtrees). The foregoing algorithm can efficiently find a heap or index page with sufficient free space, and data is then inserted into the page based on the page number.

The difficulty in managing free space lies in recycling free space. In GaussDB, there are three ways to recycle free space for Astore.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Lightweight heap page cleanup**

When a query scans an Astore heap table page, it attempts to clean up deleted old tuples on the page. As this cleanup is only incidental to the page content, only the tuple content can be cleared. The tuple pointers must be retained to prevent empty references or null pointers on the index side. A special exception is the HOT scenario, where the index key values of all indexes in the table remain unchanged before and after an update. Therefore, only a heap table tuple needs to be inserted for the updated tuple, and no index tuple needs to be inserted. If multiple tuples on a HOT chain on the same page are old enough, all intermediate tuple pointers can be deleted during cleanup, only the tuple pointer of the first version needs to be retained and redirected to the tuple pointer of the first version that does not need to be cleared.

- **Middleweight heap page and index page cleanup**

GaussDB provides the VACUUM statement for users to perform middleweight cleanup on an Astore table (or all Astore tables in a database) and its indexes. During middleweight cleanup, the query and DML operations on related tables are not blocked. In an Astore table, tuples of the old and new versions are stored together. Therefore, compared with the incidental lightweight cleanup, middleweight cleanup of an Astore table requires a full table scan or index scan to identify all old tuples to be cleaned up. Once these tuples are identified, the system first cleans them up from the index and then from the heap table, preventing the null pointer problem on the index side.

- **Heavyweight heap page and index page cleanup**

Both lightweight cleanup and middleweight cleanup can only clean up dead tuples on local Astore pages and cannot release the free space. (The released space can only be used by the table.) Therefore, GaussDB provides the VACUUM FULL statement for users to actively perform heavyweight cleanup on an Astore table (or all Astore tables in a database) and its indexes. During heavyweight cleanup, all tuples that are not dead (but may have been deleted) in a table are reinserted into a new heap table file. Then all indexes are rebuilt based on the new heap table file, thereby recycling all free space. In the main process of heavyweight cleanup, only the read-only query operation is allowed. In the commit process of heavyweight cleanup, the read-only query operation is also blocked. To improve the performance of rebuilt indexes, if indexes exist in the user heap table, index scan is used for the full table scan.

Differences Between Astore and Ustore

Both Ustore and Astore are storage engines supported by GaussDB. They share the following commonalities: both are based on a unified transaction, log, and concurrency control mechanism; both are designed to carry row-oriented workload data; and both support the recycle bin (flashback DROP/TRUNCATE) feature.

The core difference between the two lies in their implementation of multi-versioning (MVCC). Ustore uses in-place updates, where the latest and historical versions are stored separately, relying on undo rollback segments to manage historical versions. Conversely, Astore uses append-only storage, where the latest and historical versions are stored together without rollback segments. This difference stems primarily from their distinct design objectives: Ustore focuses on space controllability and rich features in update scenarios, whereas Astore focuses on simple, reliable recovery and efficient writes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Therefore, Ustore is better suited for write/delete-intensive OLTP scenarios, offering controllable space bloat and full flashback support, though updates and single-row scans incur slightly higher overhead. In contrast, Astore features a simpler exception recovery process and faster rollback, but is prone to space bloat under frequent updates.

Comparison Dimension	Ustore (In-place Update)	Astore (Append Update)
Similarity	Both are row-store engines based on a unified transaction, log, and concurrency control mechanism, and both support recycle bin and flashback DROP/TRUNCATE.	
Multi-version implementation	In-place updates; latest and historical versions are stored separately.	Append-only storage; latest and historical versions are stored together.
Rollback segment	With undo rollback segments.	No rollback segments.
Space Management	Supports reclaiming of historical versions in batches, ensuring controllable space bloat.	Prone to space bloat under frequent updates.
Exception recovery	Requires both redo and undo logs; recovery is relatively complex.	Features a simple recovery process and fast rollback.
Flashback capability	Supports flashback queries, flashback tables, and flashback DROP/TRUNCATE.	Supports only flashback DROP/TRUNCATE.

Features and operations related to storage engines

After learning the core concepts and classifications of storage engines, you can refer to the following sections to explore more details and configure the feature for your actual production workloads:

- Setting the storage engine: When creating a table, you can specify the engine type using `WITH (STORAGE_TYPE)` and use the GUC parameter `enable_default_ustore_table` to specify the default engine for row-store tables. For details, see **Setting Up a Storage Engine**.
- Querying the storage mode of a table: To view the storage engine used by an existing table, see **How Do I Query the Storage Mode of a Table?**.
- Ustore usage and best practices: To learn about the feature specifications, test methods, and best practices of Ustore, see **Ustore**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.8 Index

Description

An index is a data structure used in databases to accelerate data retrieval. By creating pointers to the data in a table, databases can quickly locate and access specific rows without scanning the entire table. Indexes function similarly to the table of contents in a book, which helps users quickly locate the desired parts and significantly improves query performance. Using indexes helps accelerate data access but can also prolong the time needed to insert, update, or delete data. Therefore, it is crucial to carefully evaluate whether to create indexes for a table and which specific columns to index.

Index Type

- B-tree: A structure similar to B+ tree. The leaf nodes store data in the sequence of key values. It is applicable to equality query, range query, sorting, and grouping.
- UB-tree: Multi-version B-tree index used only for Ustore tables. The index page contains transaction information and can be recycled. By default, the INSERTPT feature is enabled for UB-tree indexes.
- GIN: Generalized inverted index used to process indexes that are combined values, such as arrays or JSONB.
- UGIN: Generalized inverted index used only for Ustore tables.
- GiST: Suitable for the set data type and multidimensional data types such as geometric and geographic. The following data types are supported: box, point, poly, circle, tsvector, tsquery, and range.

Examples

- Single-column index

  An index created based on a single column, which is applicable to simple query conditions.

```
-- Create a table.
gaussdb=# CREATE TABLE tb_t1(c1 int, c2 int);

-- Create an index.
gaussdb=# CREATE INDEX idx_t1_c1 ON tb_t1(c1);

-- Drop the table.
gaussdb=# DROP TABLE tb_t1;
```

- Composite index

  A composite index created based on multiple columns, which is applicable to multi-condition queries.

```
-- Create a table.
gaussdb=# CREATE TABLE tb_t2(c1 int, c2 int, c3 int);

-- Create an index.
gaussdb=# CREATE INDEX idx_t2_c1_c2 ON tb_t2(c1,c2);

-- Drop the table.
gaussdb=# DROP TABLE tb_t2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Unique index

  An index created based on a unique column that allows **NULL** values. When a UNIQUE constraint is added to a table, the database automatically adds a unique index to the table.

```
-- Create a table.
gaussdb=# CREATE TABLE tb_userinfo(uid char(5), name varchar(20));

-- Create a unique index.
gaussdb=# CREATE UNIQUE INDEX idx_userinfo_uid ON tb_userinfo(uid);

-- Drop the table.
gaussdb=# DROP TABLE tb_userinfo;
```

- Expression index

  An index created based on the result of a calculation expression or function instead of the original columns in a table. It allows for fast retrieval of some calculated or converted values and is applicable when the query condition contains functions or expressions.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tb_test(c1 varchar,c2 varchar);
gaussdb=# INSERT INTO tb_test VALUES ('AAa','AAA12');
gaussdb=# INSERT INTO tb_test VALUES ('ABa','AAA13');
gaussdb=# INSERT INTO tb_test VALUES ('AAc','AAA14');

-- Create an index.
gaussdb=# CREATE INDEX idx_test_c1_lower ON tb_test(lower(c1));

-- Query.
gaussdb=# SELECT * FROM tb_test WHERE lower(c1) = 'aac';

-- Drop the table.
gaussdb=# DROP TABLE tb_test;
```

- Partial index

  An index created only for data that meets specific conditions in a table. Using such indexes can reduce the index size and improve the write performance.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tb_users(uid varchar, name varchar, is_active boolean);
gaussdb=# INSERT INTO tb_users VALUES('bc'||generate_series(1,100),'uname'||
generate_series(1,100),true);
gaussdb=# INSERT INTO tb_users VALUES('ac'||generate_series(1,50),'uname'||
generate_series(1,50),false);

-- Create an index.
gaussdb=# CREATE INDEX idx_users_uid_act ON tb_users(uid) WHERE is_active = true;

-- Partial index is automatically used during queries.
gaussdb=# SELECT * FROM tb_users WHERE uid = 'bc001' AND is_active = true;

-- Drop the table.
gaussdb=# DROP TABLE tb_users;
```

- Partitioned index

```
-- Create a table.
gaussdb=# CREATE TABLE student(id int, name varchar(20)) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Create a local partitioned index without specifying the index partition name.
gaussdb=# CREATE INDEX idx_student1 ON student(id) LOCAL;

-- Check the index partition information. It is found that the number of local index partitions is the
same as the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student1'::regclass;
  relname
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-------------
 p1_id_idx
 pmax_id_idx
(2 rows)

-- Drop the local partitioned index.
gaussdb=# DROP INDEX idx_student1;

-- Create a global index.
gaussdb=# CREATE INDEX idx_student2 ON student(id) GLOBAL;

-- Check the index partition information. It is found that the number of global index partitions is
different from the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student2'::regclass;
 relname
---------
(0 rows)

-- Drop the global index.
gaussdb=# DROP INDEX idx_student2;

-- Create a local expression index without specifying the index partition name.
gaussdb=# CREATE INDEX idx_student3 ON student(abs(id)) LOCAL;

-- Check the index partition information. It is found that the number of local index partitions is the
same as the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student3'::regclass;
   relname
-------------
 p1_abs_idx
 pmax_abs_idx
(2 rows)

-- Drop the local expression index.
gaussdb=# DROP INDEX idx_student3;

-- Create a global expression index.
gaussdb=# CREATE INDEX idx_student4 ON student(abs(id)) GLOBAL;

-- Check the index partition information. It is found that the number of global expression index
partitions is different from the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student4'::regclass;
 relname
---------
(0 rows)

-- Drop the global expression index.
gaussdb=# DROP INDEX idx_student4;

-- Drop the table.
gaussdb=# DROP TABLE student;
```

- UGIN

```
-- Create a table.
gaussdb=# CREATE TABLE test_trgm(t varchar(100)) WITH (STORAGE_TYPE=USTORE);

-- Insert data.
gaussdb=# INSERT INTO test_trgm VALUES('qwertyu0001');
gaussdb=# INSERT INTO test_trgm VALUES('qwertyu0099');
gaussdb=# INSERT INTO test_trgm VALUES ('2024 Happy New Year 7 days');

-- Create a UGIN.
gaussdb=# create index trgm_idx on test_trgm using ugin (t ugin_trgm_ops);

-- Query using the LIKE statement.
gaussdb=# select * from test_trgm where t like '%24 Happy%';

-- Drop.
DROP TABLE test_trgm;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- GIN

```
-- Create a table.
gaussdb=# CREATE TABLE test_gin (id INT, info INT[]) WITH(STORAGE_TYPE = ASTORE);

-- Create an index.
gaussdb=# CREATE INDEX test_gin_idx ON test_gin USING GIN(info);

-- Insert data.
gaussdb=# INSERT INTO test_gin SELECT g, ARRAY[1, g % 5, g] FROM generate_series(1, 20000) g;

-- Query.
gaussdb=# SELECT * FROM test_gin WHERE info @> '{2}' AND info @> '{22}' ORDER BY id, info;

-- Drop the table.
gaussdb=# DROP TABLE test_gin;
```

- GiST

```
-- Create a table.
gaussdb=# CREATE TABLE gist_box_tbl(f1 point, f2 int)WITH(storage_type = astore);

-- Create an index.
gaussdb=# CREATE INDEX gist_box_index ON gist_box_tbl USING gist(f1);

-- Insert data.
gaussdb=# INSERT INTO gist_box_tbl VALUES(point(1,2),1);
gaussdb=# INSERT INTO gist_box_tbl VALUES(point(3,4),3);

-- Query.
gaussdb=# SELECT count(*) FROM gist_box_tbl WHERE f1 ~= point(1, 2);

-- Drop the table.
gaussdb=# DROP TABLE gist_box_tbl;
```

8.9 Materialized Views

Overview

A materialized view is a special physical table that stores query results instead of query definitions. It is different from a view. Compared with materialized views, materialized views have a higher query speed. However, materialized views cannot be updated in real time and are not flexible for services that require real-time response. Materialized views are classified into fast-refresh materialized view and complete-refresh materialized view.

- Complete-refresh materialized view: When the complete-refresh materialized view needs to be refreshed, the view is fully updated. That is, the query is performed again and the result is replaced in the current view. This mode applies to scenarios where data is not frequently updated and the query cost is high. The complete-refresh materialized view update cost is high and therefore this mode is not applicable to scenarios where data is frequently updated.
- Fast-refresh materialized view: allows incremental update of the view, that is, only the data that has changed since the last refresh is updated. It applies to scenarios with high refresh frequency. Although the fast-refresh materialized view is cost-effective, it supports only a few scenarios (for example, aggregate functions are not supported). Currently, only simple filter query and base table UNION ALL are supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

- Complete-refresh materialized view

```
-- Create a table and insert data (complete-refresh materialized view supports only the Astore base table).
gaussdb=# CREATE TABLE tb_test(pid varchar(5),snum varchar(5)) WITH (storage_type = astore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'pid' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO tb_test VALUES ('00001','aaaaa');
INSERT 0 1
gaussdb=# INSERT INTO tb_test VALUES ('00001','aacba');
INSERT 0 1
gaussdb=# INSERT INTO tb_test VALUES ('00002','badbc');
INSERT 0 1

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW mv_test AS SELECT pid,count(snum) cnt FROM tb_test
GROUP BY pid;
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'pid' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
INSERT 0 2


-- Query the complete-refresh materialized view.
gaussdb=# SELECT * FROM mv_test;
  pid  | cnt
-------+-----
 00002 |   1
 00001 |   2
(2 rows)

-- Insert data, and then continue to query complete-refresh materialized view.
gaussdb=# INSERT INTO tb_test VALUES ('00002','bdacc');
INSERT 0 1
-- The query result does not change. Manually refresh the materialized view.
gaussdb=# SELECT * FROM mv_test;
  pid  | cnt
-------+-----
 00002 |   1
 00001 |   2
(2 rows)


-- Refresh the complete-refresh materialized view.
gaussdb=# REFRESH MATERIALIZED VIEW mv_test;
INSERT 0 2

-- Query the complete-refresh materialized view.
gaussdb=# SELECT * FROM mv_test;
  pid  | cnt
-------+-----
 00002 |   2
 00001 |   2
(2 rows)


-- Drop the table and materialized view.
gaussdb=# DROP MATERIALIZED VIEW mv_test;
DROP MATERIALIZED VIEW
gaussdb=# DROP TABLE tb_test;
DROP TABLE
```

- Fast-refresh materialized view

```
-- Create a table.
gaussdb=# CREATE TABLE tb_test(c1 int) WITH (storage_type = astore);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- An error is reported when the fast-refresh materialized view contains an aggregate function.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW imv_test AS SELECT count(*) FROM tb_test;
ERROR:  Feature not supported
DETAIL:  aggregates on incremental materialized view creation

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW imv_test AS SELECT * FROM tb_test;
CREATE MATERIALIZED VIEW

-- Incrementally refresh the fast-refresh materialized view.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW imv_test;
REFRESH MATERIALIZED VIEW

-- Fully refresh the fast-refresh materialized view.
gaussdb=# REFRESH MATERIALIZED VIEW imv_test;
REFRESH MATERIALIZED VIEW

-- Drop the table and materialized view.
gaussdb=# DROP MATERIALIZED VIEW imv_test;
DROP MATERIALIZED VIEW
gaussdb=# DROP TABLE tb_test;
DROP TABLE
```

8.10 Transactions

A transaction is the minimum logical unit in the execution process of the database management system. It consists of a limited database operation sequence (usually consisting of all database operations performed between the start and end of a transaction). These operations are either all successfully performed or all not performed, and are an inseparable execution unit.

When a transaction is committed, ensure that all operations in the transaction are successfully completed in the database management system. If any operation in the transaction fails, the system rolls back the database to the status before the transaction starts. The following are the four characteristics (ACID) of a transaction:

- Atomicity (A) indicates that a transaction is composed of an indivisible unit of work. Operations performed in a transaction must be all finished or have not been performed.
- Consistency (C) indicates that transactions must always keep the system in a consistent state.
- Isolation (I) ensures that concurrently executed transactions are isolated from each other and do not interfere with each other.
- Durability (D) indicates that once a transaction is committed, changes to the database must be permanently stored in the database.

Transaction Isolation Levels

- Description of dirty read, phantom read, and non-repeatable read:
  - Dirty read: The current transaction can read data that has not been committed by another transaction.
  - Phantom read: In the same transaction, when the same query is executed for multiple times, the number of rows in the result set changes due to the insert and delete operations of other transactions.
  - Non-repeatable read: In the same transaction, when the same data is read for multiple times, the read results are inconsistent due to the modification operations of other transactions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The isolation levels of GaussDB have the following features:
  - READ COMMITTED: default isolation level. The current transaction can read only the data committed by other transactions, avoiding dirty reads. However, other transactions can modify or insert data during two reads, resulting in phantom reads and non-repeatable reads.
  - REPEATABLE READ: Consistent data can be queried during the execution of the current transaction, avoiding dirty reads and non-repeatable reads.
  - SERIALIZABLE: Currently, GaussDB does not support this isolation level. Setting this isolation level is equivalent to REPEATABLE READ.

Transaction Control Statements

- START TRANSACTION: starts a transaction. If the isolation level and read/write mode are specified, the new transaction uses these features. This statement is optional.
- BEGIN: starts a transaction. If the isolation level and read/write mode are specified, the new transaction uses these features. This statement is optional.
  - If the value of the system variable *autocommit* of a session is **off**, the automatic transaction commit function is disabled. In this case, the BEGIN or START TRANSACTION statement is not required to identify a transaction consisting of multiple SQL statements.
  - If the value of the system variable *autocommit* of a session is **on**, the automatic transaction commit function is enabled. In this case, each SQL statement is an independent transaction. If multiple SQL statements are required for a transaction, run the BEGIN or START TRANSACTION command to start the transaction.
- SAVEPOINT: creates a savepoint in the current transaction. A savepoint is a special mark in a transaction. It allows all commands that are executed after the savepoint was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.
- COMMIT|END: commits and ends the current transaction, makes all modifications persistent and take effect, and clears all savepoints to release the transaction lock.
- ROLLBACK: rolls back all modifications performed in a transaction and clears all savepoints to release the transaction lock.
- ROLLBACK TO SAVEPOINT: rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Syntax

- START TRANSACTION

```
START TRANSACTION
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE
READ }
    | { READ WRITE | READ ONLY }
  } [, ...]
 ];
```

- BEGIN

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE
READ }
    | { READ WRITE | READ ONLY }
   } [, ...]
 ];
```

- SAVEPOINT
  ```
  SAVEPOINT savepoint_name;
  ```
- ROLLBACK TO SAVEPOINT
  ```
  ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
  ```

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE test1(c1 INT);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ
WRITE:
gaussdb=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
START TRANSACTION

-- Insert data.
gaussdb=# INSERT INTO test1 (c1) VALUES (1);
INSERT 0 1

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;
SAVEPOINT

-- Insert data.
gaussdb=# INSERT INTO test1 (c1) VALUES (2);
INSERT 0 1

-- Roll back to the savepoint.
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;
ROLLBACK

-- View data. You can see that the data has been rolled back to the state before the savepoint.
gaussdb=# SELECT * FROM test1;
 c1
----
  1
(1 row)
-- The isolation level is READ COMMITTED and the transaction is not complete. Therefore, a new session
will not have any data in the table.
gaussdb=# SELECT * FROM test1;
 c1
----
(0 rows)

-- Insert data.
gaussdb=# INSERT INTO test1 (c1) VALUES (3);
INSERT 0 1


-- End the transaction.
gaussdb=# COMMIT;
COMMIT

-- View data. The transaction ends, and the data can be queried in another session.
gaussdb=# SELECT * FROM test1;
 c1
----
  3
  1
(2 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
gaussdb=# DROP TABLE test1;
DROP TABLE
```

8.11 Locks

Overview

A database uses locks to control the concurrency of public resources. The general process of using a lock consists of three steps: lock addition, critical area operation, and lock release.

When a DDL/DML operation is performed on a table, the database locks the table and releases the table when the transaction ends. GaussDB provides eight levels of locks for concurrent execution of different statements. The following table lists the locks corresponding to each operation.

Table 8-1 Locks

Lock Level	Name	Usage	Conflict
1	AccessShareLock	SELECT	8
2	RowShareLock	SELECT FOR UPDATE/FOR SHARE	7,8
3	RowExclusiveLock	INSERT/UPDATE/DELETE	5,6,7,8
4	ShareUpdateExclusiveLock	VACUUM	4,5,6,7,8
5	ShareLock	CREATE INDEX	3,4,6,7,8
6	ShareRowExclusiveLock	ROW SELECT ... FOR UPDATE	3,4,5,6,7,8
7	ExclusiveLock	BLOCK ROW SHARE/ SELECT ... FOR UPDATE	2,3,4,5,6,7,8
8	AccessExclusiveLock	DROP CLASS/VACUUM FULL	1,2,3,4,5,6,7,8

When the locks of two transactions conflict, the thread that fails to obtain the lock waits for the lock. If the wait duration exceeds the value of **lockwait_timeout** (20 minutes by default), the lock wait times out. For details about other lock-related parameters, see "Database GUC Parameters > GUC Parameters > Lock Management" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Lock example:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE testl1(c1 INT, c2 VARCHAR(5));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# INSERT INTO testl1 VALUES (1,'a'),(2,'b'),(3,'c');
INSERT 0 3

-- View session parameters.
gaussdb=# SHOW lockwait_timeout;
 lockwait_timeout
------------------
 20min
(1 row)

-- Perform the operation in the first session.
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO testl1 VALUES (4,'d');
INSERT 0 1

-- Perform the operation in the second session. The SQL statement is executed only after the first session
ends.
gaussdb=# CREATE INDEX idx_testl1_c1 ON testl1(c1);

-- End the transaction in the first session.
gaussdb=# END;
COMMIT

-- Drop the table.
gaussdb=# DROP TABLE testl1;
DROP TABLE
```

Querying Lock Waits

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE testl2(id INT,info VARCHAR(10));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# INSERT INTO testl2 VALUES (1,'info1'),(2,'info2');
INSERT 0 2

-- Perform the operation in the first session.
gaussdb=# BEGIN;
BEGIN
gaussdb=# CREATE INDEX idx_testl2_id ON testl2(id);
CREATE INDEX

-- Perform the operation in the second session.
gaussdb=# INSERT INTO testl2 VALUES (3,'info3');
```

- Open a new session and run the following SQL statement to query the SQL statements to be executed:

  ```
  gaussdb=# SELECT datname,pid,query,query_id,waiting FROM pg_stat_activity WHERE waiting = TRUE;
  datname  |       pid       |                  query                  |     query_id     | waiting
  ----------+-----------------+-----------------------------------------+------------------+---------
   postgres | 140389444876032 | INSERT INTO testl2 VALUES (3,'info3'); | 3940649678789642 | t
  (1 row)
  ```

- Query wait events by PID.

  ```
  gaussdb=# SELECT node_name,db_name,query_id,tid,wait_status,wait_event FROM
  pg_thread_wait_status WHERE tid = 140389444876032 AND db_name IS NOT NULL;
   node_name | db_name  |     query_id     |       tid       | wait_status  | wait_event
  -----------+----------+------------------+-----------------+--------------+------------
   dn_6001   | postgres | 3940649678789642 | 140389444876032 | acquire lock | relation
  (1 row)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Run the following SQL statements to query PIDs of SQL statements that hold locks:

```
-- Run the following SQL statement to query PIDs of SQL statements that hold locks. The value of
locktype is the same as that of wait_event.
gaussdb=# SELECT relation,pid,mode,granted FROM pg_locks WHERE pid = 140389444876032 AND
locktype='relation';
 relation |       pid       |       mode       | granted
----------+-----------------+------------------+---------
    26466 | 140389444876032 | RowExclusiveLock | f
(1 row)

-- Run the following statement to query PIDs of SQL statements that hold locks. If the value of
granted is t, the statement is a statement that holds a lock.
gaussdb=# SELECT relation,pid,mode,granted FROM pg_locks WHERE relation = 26466;
 relation |       pid       |       mode       | granted
----------+-----------------+------------------+---------
    26466 | 140389444876032 | RowExclusiveLock | f
    26466 | 140389939934976 | AccessShareLock  | t
    26466 | 140389939934976 | ShareLock        | t
(3 rows)
```

- Run the following SQL statements to query details about SQL statements that hold locks:

  If the following information is displayed, the statement status is **idle in transaction**, indicating that the transaction is to be committed. After a transaction is manually committed, the INSERT statement starts to be executed.

```
gaussdb=# SELECT datname,pid,query,query_id,waiting,state FROM pg_stat_activity WHERE pid =
140389939934976;
 datname  |       pid       |                  query                  | query_id | waiting |        state
----------+-----------------+-----------------------------------------+----------+---------+---------------------
 postgres | 140389939934976 | CREATE INDEX idx_testl2_id ON testl2(id); |        0 | f       | idle in
transaction
(1 row)
```

  Determine whether to use the pg_terminate_backend function to end the SQL thread that holds the lock on the corresponding node according to the actual situation.

  Example:

```
gaussdb=# SELECT pg_terminate_backend(140389939934976);
 pg_terminate_backend
----------------------
 t
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE testl2;
DROP TABLE
```

8.12 PL/SQL

8.12.1 Introduction to PL/SQL

PL/SQL provides a server-side stored procedure language, which is easy to use, secure, and portable. With this language, you can create objects to access and operate database data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PL/SQL Subprogram

A PL/SQL subprogram is a named PL/SQL block that allows the caller to provide input-only, output-only, or input and output parameters. There are four types of subprograms: stored procedure, function, package, and trigger.

- **Creating a subprogram**

  The subprogram structure is as follows:

  – (Optional) Declaration

    The declaration section includes the declaration of types, variables, constants, explicit cursors, and nested subprograms. These items are local to subprograms and do not exist after the subprograms are executed.

  – (Optional) Execution

    The execution section includes assignment statements, control execution statements, and database operation statements.

  – (Optional) Exception handling

    The exception handling section includes the code for handling exceptions.

  Adding comments to a subprogram increases the readability of the subprogram. A single-line comment starts with a double minus sign (--) and ends at the end of the line. A multi-line comment starts with **/*** and ends with ***/**.

- **Subprogram execution**

  You can execute a subprogram in any of the following methods:

  – Directly call the subprogram in the application.

  – Call the subprogram in a subprogram or trigger.

  – Use the client tool.

8.12.2 Cursors

Description

A cursor is used to process result sets affected by SQL queries. These result sets can be accessed and operated row by row. A cursor points to the query result set so that each row of data can be processed separately. Cursors are classified into explicit cursors and implicit cursors.

Cursor Attributes

%FOUND: checks whether rows are retrieved by FETCH on a cursor. If a row is retrieved, **TRUE** is returned.

%NOTFOUND: works opposite to **%FOUND**. If no row is retrieved by the FETCH operation, **TRUE** is returned.

%ISOPEN: specifies whether a cursor is opened.

%ROWCOUNT: returns the number of rows affected by the cursor.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Explicit Cursors

Explicit cursors are explicitly defined and controlled by users. They are used to process SELECT queries that return multiple rows of data. Users need to explicitly define, open, use, and close cursors in PL/SQL.

- Characteristics:
  - Flexibility: Users can control the query results in a refined manner. Data can be extracted line by line and processed based on the service logic.
  - Extensibility: Explicit cursors are explicitly defined. Therefore, new processing logic can be easily modified or added.
  - Complexity: The code is more complex. You need to carefully manage the lifecycle of cursors and ensure that cursors are closed at a proper time to release resources.
  - Resource consumption: Although displaying cursors can improve performance in some cases, more system resources may be consumed in other cases. For example, frequently opening and closing cursors may increase the database overhead and affect the overall system performance.
- Application scenarios:

  Complex operations, such as line-by-line processing and data conversion, need to be performed on the result sets.

Implicit Cursors

Implicit cursors are automatically managed by the database system. When SQL statements are executed, the database automatically creates an implicit cursor to process these operations.

- Characteristics:
  - Simplicity: Cursors do not need to be explicitly defined or controlled, reducing programming workload.
  - Performance: The database is automatically managed by the database system and has good performance.
  - Lack of flexibility: Users cannot customize the attributes and behaviors of cursors.

Examples

- Explicit cursors

```
-- Perform pre-operations: creating a table and inserting data.
gaussdb=# CREATE TABLE tb_t1(c1 int,c2 int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# INSERT INTO tb_t1 VALUES (generate_series(1,10),generate_series(1,10));
INSERT 0 10

-- The following is an example of an explicit cursor:
gaussdb=# DECLARE
    v_c1 int;
    v_c2 int;
    v_res int;
    -- Define a cursor.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    CURSOR v_cur IS SELECT * FROM tb_t1;
BEGIN
    -- Open the cursor.
    OPEN v_cur;
    LOOP
        -- Use the cursor.
        FETCH v_cur INTO v_c1,v_c2;
        EXIT WHEN v_cur %notfound;
        v_res = mod(v_c1,2);

        IF v_res = 1 THEN
            UPDATE tb_t1 SET c2 = v_c1+1 WHERE c1 = v_c1;
        END IF;

    END LOOP;
    -- Close the cursor.
    CLOSE v_cur;
END;
/
ANONYMOUS BLOCK EXECUTE

-- View the table result.
gaussdb=# SELECT * FROM tb_t1;
 c1 | c2
----+----
  1 |  2
  2 |  2
  3 |  4
  4 |  4
  5 |  6
  6 |  6
  7 |  8
  8 |  8
  9 | 10
 10 | 10
(10 rows)

-- Drop the table.
gaussdb=# DROP TABLE tb_t1;
DROP TABLE
```

- Implicit cursors

```
-- Perform the pre-operation: creating tables.
gaussdb=# CREATE TABLE tb_t1(c1 int,c2 int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE TABLE tb_t2(c1 int,c2 int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'c1' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Use an implicit cursor.
gaussdb=# BEGIN
    INSERT INTO tb_t1 VALUES (generate_series(1,4),generate_series(1,4));
    RAISE INFO '% data records are inserted into the tb_t1 table',SQL%ROWCOUNT;

    INSERT INTO tb_t2 VALUES (generate_series(1,6),generate_series(1,6));
    RAISE INFO '% data records are inserted into the tb_t2 table',SQL%ROWCOUNT;

    UPDATE tb_t1 SET c2 = c2*2 WHERE c1 > 4;
    RAISE INFO '% data records are modified in the tb_t1 table',SQL%ROWCOUNT;

    UPDATE tb_t2 SET c2 = c1*2 WHERE c1 > 4;
    RAISE INFO '% data records are modified in the tb_t2 table',SQL%ROWCOUNT;
END;
/
INFO:  4 data records are inserted into the tb_t1 table.
INFO:  6 data records are inserted into the tb_t2 table.
INFO:  No record is modified in the tb_t1 table.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
INFO:  2 records are modified in the tb_t2 table.
ANONYMOUS BLOCK EXECUTE

-- View data in the tb_t1 and tb_t2 tables.
gaussdb=# SELECT * FROM tb_t1 ORDER BY 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
(4 rows)

gaussdb=# SELECT * FROM tb_t2 ORDER BY 1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  5 | 10
  6 | 12
(6 rows)

-- Drop tables.
gaussdb=# DROP TABLE tb_t1;
DROP TABLE
gaussdb=# DROP TABLE tb_t2;
DROP TABLE
```

8.12.3 Stored Procedures

Description

A stored procedure is a program that is prepared and stored in a database. It consists of a group of SQL statements and is used to perform a series of database operations. A stored procedure can be called by multiple applications, improving code reusability and maintainability.

The stored procedure has the following features:

- Complex data operation simplification: Execute complex service logic that requires multiple steps, such as batch data processing and data conversion.
- Performance optimization: Preparation and network communication reduction are used to improve the execution efficiency of database operations.
- Security hardening: Service logic is encapsulated in stored procedures to restrict direct access to database tables, hardening data security.
- Development and maintenance simplification: Service logic is integrated in the database, simplifying application development and maintenance.

Examples

In the following example, the stored procedure **prc_test** is used to insert data from the **tb_test1** table to the **tb_test2** table. If the insertion fails, the stored procedure inserts the calling time, error information, and data into the **tb_test_err** table.

```
-- Environment preparation: Create tables and insert data into the tables.
gaussdb=# CREATE TABLE tb_test1(id int,info text);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE TABLE tb_test2(id int PRIMARY KEY, info text);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tb_test2_pkey" for table "tb_test2"
CREATE TABLE
gaussdb=# INSERT INTO tb_test1 VALUES (generate_series(1,5),'AAA');
INSERT 0 5
gaussdb=# INSERT INTO tb_test2 VALUES (generate_series(1,2),'aaa');
INSERT 0 2
gaussdb=# CREATE TABLE tb_test_err(
    err_time text,
    err_info text,
    id int,
    info text
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'err_time' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Create the stored procedure prc_test.
gaussdb=# CREATE OR REPLACE PROCEDURE prc_test(v_time IN text) AS
    -- Define a cursor.
    CURSOR v_cur IS SELECT * FROM tb_test1;
    v_id int;
    v_info text;
BEGIN
    -- Open the cursor.
    OPEN v_cur;
    LOOP
        -- Use the cursor.
        FETCH v_cur INTO v_id,v_info;
        -- Loop exit condition: The cursor cannot obtain data.
        EXIT WHEN v_cur %notfound;
        BEGIN
            -- Insert data from the tb_test1 table to the tb_test2 table.
            INSERT INTO tb_test2 VALUES (v_id,v_info);
        -- Troubleshooting: If the execution fails, insert the error information to the tb_test_err table.
        EXCEPTION
            WHEN OTHERS THEN
                INSERT INTO tb_test_err VALUES (v_time,sqlerrm,v_id,v_info);
        END;
    END LOOP;
    -- Close the cursor.
    CLOSE v_cur;
END;
/
CREATE PROCEDURE

-- Call the stored procedure to pass the character string 2019-01-31.
gaussdb=# CALL prc_test('2019-01-30');

-- View the result.
gaussdb=# SELECT * FROM tb_test2 ORDER BY id;
 id | info
----+------
  1 | aaa
  2 | aaa
  3 | AAA
  4 | AAA
  5 | AAA
(5 rows)

-- View the data that fails to be inserted.
gaussdb=# SELECT * FROM tb_test_err;
  err_time  |                          err_info                          | id | info
------------+------------------------------------------------------------+----+------
 2019-01-30 | Duplicate key value violates unique constraint "tb_test2_pkey". |  1 | AAA
 2019-01-30 | Duplicate key value violates unique constraint "tb_test2_pkey". |  2 | AAA
(2 rows)

-- Drop the table and stored procedure.
gaussdb=# DROP PROCEDURE prc_test;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
DROP PROCEDURE
gaussdb=# DROP TABLE tb_test1,tb_test2,tb_test_err;
DROP TABLE
```

8.12.4 Functions

Description

A function is a database object that returns a single value or a table object. A function must have a return value and the type of the return value must be defined. It applies to scenarios where a single value needs to be returned, for example, calculation, data conversion, or table data return.

Examples

In the following example, the function add_num() is created. The input parameter is two integer parameters, and the sum of them is returned.

```
-- Create a function.
gaussdb=# CREATE OR REPLACE FUNCTION add_num(int, int) RETURNS int AS $$
BEGIN
       RETURN $1+$2;
END;
$$ LANGUAGE plpgsql;

-- Use the add_num() function.
gaussdb=# SELECT add_num(3,4);
 add_num
---------
       7
(1 row)

-- Drop the function.
gaussdb=# DROP FUNCTION add_num;
DROP FUNCTION
```

8.12.5 Trigger

Description

A trigger is a special function that is automatically executed when a data change event (INSERT, UPDATE, DELETE, or TRUNCATE) or a database event (DDL statement) occurs. It is not called by users or applications.

Table 8-2 Types of triggers supported on tables and views

Trigger Time	Trigger Event	Row-Level	Statement-Level
BEFORE	INSERT/UPDATE/ DELETE	Tables	Tables and views
	TRUNCATE	Not supported	Tables
AFTER	INSERT/UPDATE/ DELETE	Tables	Tables and views
	TRUNCATE	Not supported	Tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Trigger Time	Trigger Event	Row-Level	Statement-Level
INSTEAD OF	INSERT/UPDATE/ DELETE	Views	Not supported
	TRUNCATE	Not supported	Not supported

Managing Triggers

- Create a trigger.

```
-- Create the company table.
gaussdb=# CREATE TABLE company(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "company_pkey" for table
"company"
CREATE TABLE

-- Create the audit table.
gaussdb=# CREATE TABLE audit(
   EMP_ID INT NOT NULL,
   ENTRY_DATE TEXT NOT NULL);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'emp_id' as the distribution column by
default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Use the CREATE FUNCTION statement to create a trigger function.
gaussdb=#  CREATE OR REPLACE FUNCTION auditlogfunc() RETURNS TRIGGER AS $example_table$
   BEGIN
      INSERT INTO AUDIT(EMP_ID, ENTRY_DATE) VALUES (new.ID, current_timestamp);
      RETURN NEW;
   END;
$example_table$ LANGUAGE plpgsql;
CREATE FUNCTION

-- Use the CREATE TRIGGER statement to join the function with a table.
gaussdb=# CREATE TRIGGER example_trigger AFTER INSERT ON company FOR EACH ROW EXECUTE
PROCEDURE auditlogfunc();
CREATE TRIGGER

-- Insert data to the COMPANY table.
gaussdb=# INSERT INTO company (ID,NAME,AGE,ADDRESS,SALARY) VALUES (1, 'Zhang San', 32,
'Jinan', 20000.00 );
INSERT 0 1

-- View data in the AUDIT table.
gaussdb=# SELECT * FROM audit;
 emp_id |          entry_date
--------+-------------------------------
      1 | 2024-12-09 17:19:54.70118+08
(1 row)
```

- Modify a trigger.

  GaussDB provides the ALTER TRIGGER statement to modify a trigger.

```
-- Rename a trigger.
gaussdb=# ALTER TRIGGER example_trigger ON company RENAME TO example_trigger1;
ALTER TRIGGER
```

  By default, a trigger is enabled. You can run the following statements to disable or enable a trigger:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Disable a trigger.
gaussdb=# ALTER TABLE company DISABLE TRIGGER example_trigger1;
ALTER TABLE

-- Insert data into the COMPANY table and view the AUDIT table.
gaussdb=# INSERT INTO company (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'Li Si', 30, 'Anhui',
30000.00 );
INSERT 0 1

gaussdb=# SELECT * FROM audit;
 emp_id |          entry_date
--------+-------------------------------
      1 | 2024-12-09 11:25:55.992644+08
(1 row)

-- Enable a trigger.
gaussdb=# ALTER TABLE company ENABLE TRIGGER example_trigger1 ;
ALTER TABLE
```

- View a trigger.

  The information_schema.triggers view stores information about triggers.

```
-- Run the following meta-command to display the query result in a vertical line:
gaussdb=# \x
Expanded display is on.

gaussdb=# SELECT * FROM information_schema.triggers WHERE trigger_name = 'example_trigger1';
-[ RECORD 1 ]--------------+---------------------------------
trigger_catalog            | postgres
trigger_schema             | public
trigger_name               | example_trigger1
event_manipulation         | INSERT
event_object_catalog       | postgres
event_object_schema        | public
event_object_table         | company
action_order               |
action_condition           |
action_statement           | EXECUTE PROCEDURE auditlogfunc()
action_orientation         | ROW
action_timing              | AFTER
action_reference_old_table |
action_reference_new_table |
action_reference_old_row   |
action_reference_new_row   |
created                    |

gaussdb=# \x
Expanded display is off.
```

- Drop a trigger.

```
gaussdb=# DROP TRIGGER example_trigger1 ON company;
DROP TRIGGER

-- Drop tables.
gaussdb=# DROP TABLE company;
DROP TABLE
gaussdb=# DROP TABLE audit;
DROP TABLE
```

8.12.6 Packages

A package is a group of related subprograms, including cursors and variables used by the package. A package is stored in the database and can be used as a unit. Packaged subprograms can be explicitly called by applications or users. It has the following features:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Modular programming: Related functions and stored procedures are encapsulated in a package to facilitate modular programming. Modularization improves code readability and maintainability.
- Dependency chain disconnection: Modifying a database object (for example, modifying the table definition) does not cause all other objects (such as stored procedures, functions, and views) that depend on the object to become invalid. This avoids high-cost recompilation.
- Performance optimization: Packages are stored in the database in compilation mode. When a package is called, the database does not need to compile code again, improving the execution efficiency.
- Namespace management: A package provides a namespace to avoid naming conflicts between different modules. In the same package, there can be different functions and procedures. Even if they have the same name in other contexts, they do not conflict with each other.
- Security: Packages can be used to better control access to database objects.

Creating a Package

Creating a package requires the package header and body. The header needs to declare all public structures, and the body needs to define all structures of the package, including the public part and private part.

In the following example, the **emp** table and **pkg_emp_mgr** package are created. The package contains three stored procedures.

- **pkg_emp_mgr.emp_add** inserts employee information into the **emp** table.
- **pkg_emp_mgr.emp_del** deletes employee information from the **emp** table.
- **pkg_emp_mgr.emp_res** modifies information in the **emp** table.

```
-- PACKAGE can be used only in an ORA-compatible database.
gaussdb=# CREATE DATABASE ora_testdb DBCOMPATIBILITY 'ORA';
CREATE DATABASE

gaussdb=# \c ora_testdb
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora_testdb" as user "omm".
-- Create the emp table.
ora_testdb=# CREATE TABLE emp(
    id INT,             -- Employee ID
    name VARCHAR(50),   -- Employee name
    depno INT,          -- Department ID
    sal FLOAT           -- Salary
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Create a package header.
ora_testdb=# CREATE OR REPLACE PACKAGE pkg_emp_mgr AS
    PROCEDURE emp_add(v_id INT, v_name VARCHAR, v_depno INT, v_sal FLOAT);
    PROCEDURE emp_del(v_id INT);
    PROCEDURE emp_res(v_id INT, v_depno INT, v_sal VARCHAR);
END pkg_emp_mgr;
/

-- Create a package body.
ora_testdb=# CREATE OR REPLACE PACKAGE BODY pkg_emp_mgr AS
    PROCEDURE emp_add(v_id INT, v_name VARCHAR, v_depno INT, v_sal FLOAT) AS
    BEGIN
        INSERT INTO emp VALUES (v_id, v_name, v_depno, v_sal);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    END;
    PROCEDURE emp_del(v_id INT) AS
    BEGIN
        DELETE FROM emp WHERE id = v_id;
    END;
    PROCEDURE emp_res(v_id INT, v_depno INT, v_sal VARCHAR) AS
    BEGIN
        UPDATE emp SET depno = v_depno, sal = v_sal WHERE id = v_id;
    END;
END pkg_emp_mgr;
/
-- Call pkg_emp_mgr.emp_add.
ora_testdb=# CALL pkg_emp_mgr.emp_add(1, 'scott', 10, 4000.00);
ora_testdb=# CALL pkg_emp_mgr.emp_add(2, 'july', 20, 3000.00);
-- Query table data.
ora_testdb=# SELECT * FROM emp;
 id | name  | depno | sal
----+-------+-------+------
  1 | scott |    10 | 4000
  2 | july  |    20 | 3000
(2 rows)

-- Call pkg_emp_mgr.emp_res.
ora_testdb=# CALL pkg_emp_mgr.emp_res(2, 30, 6000.00);
-- Query data.
ora_testdb=# SELECT * FROM emp;
 id | name  | depno | sal
----+-------+-------+------
  1 | scott |    10 | 4000
  2 | july  |    30 | 6000
(2 rows)

-- Call pkg_emp_mgr.emp_del.
ora_testdb=# CALL pkg_emp_mgr.emp_del(2);
-- Query data.
ora_testdb=# SELECT * FROM emp;
 id | name  | depno | sal
----+-------+-------+------
  1 | scott |    10 | 4000
(1 row)
-- Drop the package.
ora_testdb=# DROP PACKAGE pkg_emp_mgr;
NOTICE:  drop cascades to 3 other objects
DETAIL:  drop cascades to function public.emp_add(integer,character varying,integer,double precision)
drop cascades to function public.emp_del(integer)
drop cascades to function public.emp_res(integer,integer,character varying)
DROP PACKAGE

-- Drop the table.
ora_testdb=# DROP TABLE emp;
DROP TABLE

-- Drop the database.
ora_testdb=# \c postgres
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "omm".

gaussdb=# DROP DATABASE ora_testdb;
DROP DATABASE
```

8.13 Flashback

Flashback is a core part of the database restoration technology, enabling users to selectively undo a committed transaction and recover data from erroneous manual operations. Based on the restoration objects and underlying mechanisms,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

GaussDB flashback operations are mainly categorized into flashback query, table flashback, flashback DROP/TRUNCATE, and database flashback.

Prior to the adoption of flashback technology, retrieving committed database modifications relied on backup recovery or PITR, taking minutes or even hours. With flashback technology, restoring committed transactions via Flashback DROP/TRUNCATE takes only seconds, regardless of database size.

NOTE

- Ustore supports flashback query, table flashback, flashback DROP/TRUNCATE, and database flashback, while Astore supports only flashback DROP/TRUNCATE.
- Standby nodes do not support flashback.
- You can enable flashback as required, but this will cause performance deterioration.

Why Is Flashback Needed?

During routine O&M and database development, human error represents an inevitable risk. For instance, an operator might delete a table by mistake, mistakenly execute a TRUNCATE command to purge data, or incorrectly update critical records during a service logic modification. This type of logical corruption typically affects only a single table or a specific group of tables rather than the entire database instance.

The core value of flashback becomes apparent when such logical corruption occurs. Instead of performing a full-instance backup restoration or rebuilding the entire database via PITR, DBAs can execute rapid, localized restorations specifically targeting the damaged tables or databases. This localized approach significantly minimizes the risk of service interruption caused by manual misoperations.

Advantages

- **Second-level restoration**: Reclaiming data from committed DROP or TRUNCATE operations takes only seconds, and this restoration time remains completely independent of the database size. Compared to traditional backup restoration and PITR, flashback technology drastically shortens the restoration window.
- **Precise localized restoration**: Restoring only the affected tables or objects to a specific point in time eliminates the need to roll back the entire database instance. This approach prevents other normal service data from being adversely affected when logical corruption is confined to a subset of tables.
- **Multiple restoration granularities**: A comprehensive suite of capabilities, including flashback query, table flashback, flashback DROP/TRUNCATE, and database flashback, are provided to meet a wide range of restoration needs, from simply viewing historical data to fully restoring deleted database entities.
- **Time point and CSN-based locating**: A flashback point can be specified by either timestamp or commit sequence number (CSN) to accurately locate a consistency point, meeting diversified restoration requirements.

Application Scenarios

- **Quick retrieval of tables deleted or cleared by mistake**: For development or O&M personnel who mistakenly execute a DROP or TRUNCATE command

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

during a database change, resulting in data loss from critical service tables. Provided that the recycle bin is enabled, users can use flashback DROP or flashback TRUNCATE to instantly restore the deleted table along with its associated structures, such as indexes and constraints. This approach ensures second-level service restoration.

- **Logical reconstruction of incorrectly updated data**: For service teams that incorrectly modify key records during batch updates or script executions, where the scope of logical corruption is confined to specific tables. Provided that the **undo_retention_time** parameter is configured to retain the required window of old Undo versions, users can first perform a flashback query to inspect the table's snapshot data at a specific point before the error occurred. After verifying the correct version, they can then use table flashback to restore the entire table to that exact timestamp, achieving table-level data restoration without affecting other database objects.
- **Historical data audit and comparison**: For data analysts or auditors who need to examine the state of a table at a specific point in the past to verify data modifications. Provided that the **undo_retention_time** parameter has preserved the corresponding historical versions, users can employ flashback queries to directly read historical records tied to a specific timestamp or CSN. This capability allows seamless data backtracking and logical reconstruction without setting up a separate historical database.

Working Principles

From an architectural perspective, flashback in GaussDB is implemented based on two underlying mechanisms: the Undo multi-version mechanism of Ustore and the recycle bin mechanism. The former is responsible for retaining historical versions of row data to support flashback query and table flashback operations, while the latter is responsible for saving the physical files of deleted objects to support flashback DROP, flashback TRUNCATE, and database flashback operations.

Undo-based flashback (encompassing flashback query and table flashback) relies on MVCC. When data is updated or deleted within Ustore, the Undo subsystem saves the older version separately, with its retention period controlled by the GUC parameter **undo_retention_time**. During a flashback query, the system retrieves and returns the corresponding historical data based on the specified timestamp or CSN. During a table flashback operation, building on this retrieval mechanism, the system deletes the incremental data generated after the specified time point and rolls back the deleted rows within that time window, thereby restoring the entire table to the specified historical point.

Recycle bin-based flashback (encompassing flashback DROP, flashback TRUNCATE, and database flashback) relies on the recycle bin mechanism. Once the recycle bin is enabled, when a table or index is dropped or truncated, the system automatically generates a hidden object prefixed with "BIN" in the schema of the original object, recording the physical storage location of the deleted entity. During a flashback operation, the physical files are reconstructed based on these records to restore the dropped or truncated objects. The retention period for these objects is controlled by parameters such as **recyclebin_retention_time** (at the table level), and they are automatically purged once the retention window expires. Specifically, database flashback depends on an independent database-level recycle bin (**gs_db_recyclebin**), which is controlled by parameters such as **enable_db_recyclebin** and **dbrecyclebin_retention_time**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Through the seamless collaboration of the Undo subsystem and the recycle bin mechanism, flashback can both read and restore row-level historical versions and successfully retrieve database objects that have been completely deleted.

Flashback Classification

Based on the differences in restoration objects and underlying mechanisms, flashback in GaussDB is classified into the following types:

- **Flashback Query**
  - **Overview**

    This feature allows users to query the snapshot data of a table at a specific point in the past, enabling them to view and logically reconstruct damaged data that has been accidentally deleted or modified. Based on the MVCC mechanism, the system retrieves specified historical versions by searching through historical Undo records.
  - **Prerequisites**

    Set the GUC parameter **undo_retention_time** for old undo logs.
  - **Examples**

```
-- Create a table and insert data into the table in advance.
gaussdb=# CREATE TABLE tb_t1 (c1 int, c2 int) WITH(storage_type = ustore);
gaussdb=# INSERT INTO tb_t1 values (1,1),(2,2);

-- Query the current time and CSN.
gaussdb=# SELECT now(), int8in(xidout(next_csn)) FROM gs_get_next_xid_csn() limit 1;
              now              | int8in
-------------------------------+---------
 2025-04-22 15:22:47.574995+08 | 2020912
(1 row)

-- Simulate a data misoperation.
gaussdb=# UPDATE tb_t1 SET C1=222,C2=222 WHERE C1 = 2;
gaussdb=# SELECT * FROM tb_t1;
 c1  | c2
-----+-----
   1 |   1
 222 | 222
(2 rows)

-- Query the pre-modification data in the table based on time.
gaussdb=# SELECT * FROM tb_t1 TIMECAPSULE TIMESTAMP '2025-04-22 15:22:47.574995';
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Query the pre-modification data in the table based on the CSN.
gaussdb=# SELECT * FROM tb_t1 TIMECAPSULE CSN '2020912';
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE tb_t1;
```

- **Table Flashback**
  - **Overview**

    This feature allows users to restore a table to a specific point in time or a particular CSN. When logical corruption is confined to a single table or a

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

specific group of tables rather than the entire database, this feature (based on the MVCC mechanism) enables rapid, table-level data restoration.

- **Prerequisites**

  Set the GUC parameter **undo_retention_time** for old undo logs.

- **Examples**

```
-- Create a table and insert data into the table in advance.
gaussdb=# CREATE TABLE tb_t2 (c1 int, c2 int) WITH(storage_type = ustore);
gaussdb=# INSERT INTO tb_t2 VALUES (1,1),(2,2);

-- Query the current time and CSN.
gaussdb=# SELECT now(), int8in(xidout(next_csn)) FROM gs_get_next_xid_csn() limit 1;
              now              | int8in
-------------------------------+---------
 2025-04-22 15:32:22.99132+08 | 2021175
(1 row)


-- Simulate a data misoperation.
gaussdb=# UPDATE tb_t2 SET c1 = 222, c2 = 222 WHERE C1 = 2;
gaussdb=# SELECT * FROM tb_t2;
 c1  | c2
-----+-----
   1 |   1
 222 | 222
(2 rows)

-- Use flashback to restore the table data to the time point before the misoperation.
gaussdb=# TIMECAPSULE table tb_t2 TO TIMESTAMP to_timestamp('2025-04-22
15:32:22.99132','YYYY-MM-DD HH24:MI:SS.FF');
gaussdb=# SELECT * FROM tb_t2;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- You can also run the following SQL statements to flash back the table based on the CSN:
gaussdb=# UPDATE tb_t2 SET c1 = 333, c2 = 333 where c1 = 2;
gaussdb=# TIMECAPSULE table tb_t2 TO CSN 2021175;
gaussdb=# SELECT * FROM tb_t2;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE tb_t2;
```

- **Flashback DROP/TRUNCATE**
  - **Overview**
    - Flashback DROP restores tables that are accidentally dropped, along with their structures, such as indexes and table constraints, from the recycle bin. Flashback DROP is based on the recycle bin mechanism. Users can restore physical table files from the recycle bin to restore the dropped tables.
    - Flashback TRUNCATE restores tables that are mistakenly or accidentally truncated, along with their indexes, from the recycle bin. Flashback TRUNCATE is based on the recycle bin mechanism. Users can restore physical table files from the recycle bin to restore the truncated tables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **Prerequisites**
    - Set the GUC parameter **enable_recyclebin** to **on**.
    - Set the GUC parameter **recyclebin_retention_time** to specify how long files will be kept in the recycle bin. Files will be automatically deleted from the recycle bin after this length of time.
  - **Examples**

```
-- Create a table and insert data into the table in advance.
gaussdb=# CREATE TABLE tb_t3 (c1 int,c2 int) WITH(storage_type = ustore);
gaussdb=# INSERT INTO tb_t3 VALUES (1,1), (2,2);

-- Drop the table.
gaussdb=# DROP TABLE tb_t3;

-- Flash back to the time before the table is deleted.
gaussdb=# TIMECAPSULE TABLE tb_t3 TO BEFORE DROP;
gaussdb=# SELECT * FROM tb_t3;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Use TRUNCATE to empty the table.
gaussdb=# TRUNCATE TABLE tb_t3;

-- Flash back to the time before the table is emptied.
gaussdb=# TIMECAPSULE TABLE tb_t3 TO BEFORE TRUNCATE;
gaussdb=# SELECT * FROM tb_t3;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Drop the table.
DROP TABLE tb_t3;
```

- **Database Flashback**
  - **Overview**

    This feature allows users to restore a database that is deleted by mistake and its associated objects from the database-level recycle bin. Database flashback is based on the recycle bin mechanism. You can restore physical table files recorded in the recycle bin to restore dropped databases.
  - **Prerequisites**
    - Set **enable_db_recyclebin** to enable the recycle bin.
    - Set **dbrecyclebin_retention_time** to specify the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.
    - Set **dbrecyclebin_maxsize** to specify the maximum size of the recycle bin. If the size of the recycle bin exceeds the maximum size, objects cannot be added to the recycle bin.
  - **Examples**

    After the database-level recycle bin is enabled, flash back the database in the recycle bin.

```
-- Check the recycle bin.
gaussdb=# SELECT * FROM gs_db_recyclebin;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 rcydbid | rcyname | rcyoriginname | rcyrecyclecsn | rcyrecycletime | rcytablespaces | rcydbsize
---------+---------+---------------+---------------+----------------+----------------+-----------
(0 rows)
-- Create a database.
gaussdb=# CREATE DATABASE flashback_db;
CREATE DATABASE
-- Drop the database to the recycle bin.
gaussdb=# DROP DATABASE flashback_db;
DROP DATABASE
gaussdb=# SELECT * FROM gs_db_recyclebin;
 rcydbid |          rcyname           | rcyoriginname | rcyrecyclecsn |        rcyrecycletime        |
rcytablespaces | rcydbsize
---------+----------------------------+---------------+---------------+------------------------------
+----------------+-----------
   82523 | BIN$1425B67F$2F7B1738==$0 | flashback_db  |         13916 | 2024-12-11
09:02:35.20606-05 | 1663           |     26272
(1 row)
-- Flash back the dropped database by using the recycle bin name.
gaussdb=# TIMECAPSULE DATABASE "BIN$1425B67F$2F7B1738==$0" to before drop;
TimeCapsule Database
-- Flash back the dropped database by using the original name.
gaussdb=# TIMECAPSULE DATABASE flashback_db to before drop;
TimeCapsule Database
-- Rename the database after flashback by using the recycle bin name.
gaussdb=# TIMECAPSULE DATABASE "BIN$1425B67F$2F7B1738==$0" to before drop rename to
flashback_db1;
TimeCapsule Database
-- Rename the database after flashback by using the original name.
gaussdb=# TIMECAPSULE DATABASE flashback_db to before drop rename to flashback_db1;
TimeCapsule Database
gaussdb=# SELECT * FROM gs_db_recyclebin;
 rcydbid | rcyname | rcyoriginname | rcyrecyclecsn | rcyrecycletime | rcytablespaces | rcydbsize
---------+---------+---------------+---------------+----------------+----------------+-----------
(0 rows)
-- View the database after flashback.
gaussdb=# SELECT * FROM pg_database where datname = 'flashback_db';
   datname    | datdba | encoding | datcollate  |  datctype   | datistemplate | datallowconn |
datconnlimit | datlastsysoid | datfrozenxid | dattablespace
| datcompatibility | datacl | datfrozenxid64 | datminmxid | dattimezone | dattype
--------------+--------+----------+-------------+-------------+---------------+--------------+--------------
+---------------+--------------+---------------
+------------------+--------+----------------+------------+-------------+---------
 flashback_db |     10 |        7 | en_US.UTF-8 | en_US.UTF-8 | f             | t            |           -1
|         12995 | 0            |          1663
| A                |        |          40936 |          2 | PRC         | D
(1 row)
```

Differences Between Flashback and Backup Restoration

Both flashback and backup restoration serve as critical data restoration methods in GaussDB. Their common purpose is to revert data to its pre-error state following data loss or corruption, thereby ensuring comprehensive data security and reliability.

The core difference between the two methods lies in their data sources and restoration mechanisms. Flashback operates online, leveraging historical Undo versions and recycle bin data retained directly within the active instance, enabling data restoration without dependencies on external files. Backup restoration, conversely, relies on independently stored physical backup sets (comprising full and incremental backups) as well as archived WALs (utilized for PITR), requiring the reloading of backups and the replaying of logs to reconstruct the data set. This operational divergence primarily stems from their distinct data sources: flashback depends on online, time-bound historical versions and recycle bin entities, whereas backup restoration depends on completely offline, securely stored historical backups and logs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Due to these differences, each method has its distinct strengths regarding restoration granularity, speed, and application scope. Flashback is optimized for localized restoration at the table or database level and boasts exceptional restoration speeds. Specifically, flashback Drop and flashback Truncate execute within seconds, and their restoration time remains entirely independent of the database size; however, this capability is strictly constrained to the configured Undo or recycle bin retention windows. Therefore, flashback is best suited for the rapid remediation of human errors within defined retention windows, while backup restoration is ideal for restoration scenarios involving expired retention periods, full-database disasters, or media failures. The two methods effectively complement each other.

Comparison Dimension	Flashback	Backup Restoration
Similarity	Both serve as data restoration methods used to revert data to its pre-error state.	
Data source	Historical undo versions and recycle bin within the instance.	Independent physical backups and archived WAL logs (PITR).
Restoration granularity	Localized objects at the table or database level.	The entire instance or database.
Restoration speed	Fast; flashback Drop/Truncate takes only seconds and is independent of database size.	Several minutes to several hours, depending on the data volume.
Time window	Strictly limited to the configured undo or recycle bin retention periods.	Capable of reverting to earlier historical points.
Typical scenarios	Restoration of misoperations within the retention windows.	Media corruption, instance-level disasters, and restoration beyond retention periods.

Features and Operations Related to Flashback

After learning about the core concepts and application scenarios of flashback, you can refer to the following sections for more details and configuration instructions in actual service scenarios.

- Preconfiguration switches and parameter settings:
  - Enabling the flashback function: Flashback query and table flashback depend on the retention of historical undo versions, which requires configuring **undo_retention_time**. When this function is disabled (set to **0**), the flashback function is unavailable. For details, see "Database GUC Parameters > GUC Parameters > Flashback Parameters" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Enabling the recycle bin: To use flashback DROP/TRUNCATE, you need to set **enable_recyclebin** to **on**. To use database flashback, you need to set **enable_db_recyclebin** to **on**. You can also set **recyclebin_retention_time** and **dbrecyclebin_retention_time** to specify the retention period of objects in the recycle bin. For details, see "Database GUC Parameters > GUC Parameters > Flashback Parameters" in *Reference*.
- Performing a flashback:
  - Flashback query: You can use the TIMECAPSULE { TIMESTAMP | CSN } expression to query the historical data of a table at a specified time point or CSN. For details, see **Flashback Query**.
  - Table flashback: You can use the TIMECAPSULE TABLE table_name TO { TIMESTAMP | CSN } expression to restore a table to a specified time point. For details, see **Flashback Table**.
  - Flashback DROP/TRUNCATE: You can restore a dropped or truncated table and its auxiliary structures from the recycle bin. For details, see **Flashback DROP/TRUNCATE**.

8.14 VACUUM

In GaussDB, when data is updated or deleted, the system does not immediately release the physical space; instead, it marks the space as "reusable." Over time, dead tuples accumulate in the table. Dead tuples refer to historical data rows that have been deleted or overwritten by newer versions due to updates, and are no longer referenced by any active transactions.

This not only reduces storage space utilization but also impairs query performance, as queries must traverse through these obsolete tuples. The primary purpose of the VACUUM command is to clean up these dead tuples and reclaim the corresponding storage space.

Why VACUUM Is Needed?

In summary, the core purpose of VACUUM is to clear the continuously accumulating dead tuples and reclaim the reusable space they occupy. This suppresses Astore table bloat, thereby safeguarding both space utilization and query performance.

Astore utilizes Multi-Version Concurrency Control (MVCC) to ensure data consistency and accessibility in high-concurrency environments. Every transaction runs on its own database snapshot captured at its start time, meaning that obsolete data cannot be deleted immediately. During UPDATE and DELETE operations, Astore retains historical versions of data for other active transactions. For deletions, tuples are not physically removed from data pages; instead, they are merely marked as deleted. These marked tuples are what we call "dead tuples." For updates, a similar marking mechanism is used, effectively marking the original data row as deleted before inserting a new one. Since data is not physically purged, these dead tuples continue to occupy disk space. This leads to scenarios where a table consumes a massive amount of disk space despite containing very little active data. Furthermore, because Astore's update mechanism is append-based, old and new versions of data coexist on the same data pages, further compounding database tablespace bloat.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

It should be noted that for Ustore, the routine recycling of old versions is handled autonomously by the Undo subsystem, independent of VACUUM. However, Ustore still supports VACUUM and VACUUM FULL (e.g., to completely reclaim physical space), and system catalog cleanup along with Free Space Map (FSM) maintenance still rely on autovacuum.

Advantages

- **Table bloat control**: By clearing dead tuples and maintaining the free space map (FSM), VACUUM marks the space occupied by obsolete data as reusable for subsequent inserts. This prevents continuous tablespace expansion and maximizes space recycling efficiency.
- **Lightweight and non-blocking**: A standard VACUUM operation acquires low-level locks, meaning it does not block normal read/write operations from concurrent workloads. With minimal impact on database performance, it can be safely executed while services are online.
- **Old transaction freezing**: During cleanup, it attempts to freeze older transactions, making historical tuples visible to all snapshots. This accelerates visibility checks and pre-modification validation, facilitates the advancement of the table's frozen transaction ID (relfrozenxid), and enables the recycling of transaction log files such as CLOG and CSNLOG.
- **Automatic cleanup**: Through the autovacuum daemon, the database automatically triggers cleanup and statistics updates once dead tuples cross designated thresholds, significantly easing the burden of manual O&M.

Application Scenarios

- **Managing bloat in frequently updated tables**: For Astore service tables with high-frequency UPDATE or DELETE operations, such as order or accounting tables, O&M teams often struggle with severe table bloat where the consumed physical space far exceeds the size of active data. Periodically running VACUUM on these tables clears dead tuples and marks their slots as reusable. This approach effectively controls table bloat and maintains stable query performance. Furthermore, it operates with minimal locking and does not block routine read/write workloads.
- **Completely reclaiming disk space**: For service tables that have already suffered massive bloat and require their physical disk space to be genuinely returned to the OS, O&M teams can execute VACUUM FULL during off-peak windows. This operation completely rebuilds the original table and physically releases the unneeded space. However, it requires an exclusive lock on the table, preventing any read or write access during its execution. Consequently, this command is best reserved for scheduled maintenance windows or deep off-peak hours to reorganize large tables.
- **Automating routine maintenance**: For routine O&M operations that require no manual intervention, DBAs can rely on enabling autovacuum. This background daemon automatically monitors the accumulation of dead tuples across all tables. Once a designated threshold is crossed, it automatically triggers VACUUM and updates the optimizer statistics. This continuous automation ensures that both table bloat and stale statistics are constantly kept in check during daily production runs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Working Principles

The execution of VACUUM can be summarized into four key phases: "scanning and cleanup, space maintenance, transaction freezing, and information update." The system begins by scanning the data pages within the table. Under the default Lazy mode, it leverages the Visibility Map (VM) to skip fully visible pages, marking the encountered dead tuples as reusable. However, a standard VACUUM operation does not reorganize the physical storage layouts of active tuples, nor does it immediately return the physical space to the OS. Instead, it maintains the FSM to record these available slots for future insertion and reuse.

Building on this cleanup process, VACUUM attempts to freeze older transactions, marking tuple transaction states as visible to all snapshots to accelerate subsequent visibility checks. Concurrently, it updates the VM, refreshes optimizer statistics, and reclaims obsolete transaction log files. This way, a standard VACUUM operation acquires only low-level locks, ensuring that normal read/write workloads from active services remain entirely unblocked.

Automatic cleanup is handled by the autovacuum daemon, which orchestrates two types of threads: the launcher and the worker. The autovacuum launcher remains resident as a background process when the feature is enabled, taking charge of scheduling. The autovacuum worker threads dynamically connect to specific databases to execute the actual VACUUM operations on selected tables. This automation is driven by a dead tuple threshold. Specifically, the autovacuum process is triggered when the number of dead tuples exceeds the following formula: autovacuum_vacuum_scale_factor × table row count + autovacuum_vacuum_threshold. By combining manual VACUUM commands with the automated autovacuum daemon, the database continuously reclaims space and maintains fresh statistics over long-term operations, thereby guaranteeing sustained and stable performance.

VACUUM Classification

Based on cleanup intensity, triggering modes, and target objects, VACUUM operations can be classified into several categories:

1. By cleanup intensity, they are divided into standard VACUUM and VACUUM FULL. A standard VACUUM only marks the space of dead tuples as reusable, offering a lightweight and non-blocking cleanup. Conversely, VACUUM FULL completely rebuilds the table and returns physical space to the OS; however, it requires an exclusive lock on the table, making it far more thorough but significantly more expensive.
   - **VACUUM**
     - Examples

```
-- Create the tbl_test table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(c1 int);
gaussdb=# INSERT INTO tbl_test VALUES (1);

-- View data and the ctid of the data.
gaussdb=# SELECT ctid,* FROM tbl_test;
 ctid  | c1
-------+----
 (0,1) |  1
(1 row)

-- Delete data.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# DELETE FROM tbl_test;

-- Insert a data record again. A new ctid is used.
gaussdb=# INSERT INTO tbl_test VALUES (2);
gaussdb=# SELECT ctid,* FROM tbl_test;
 ctid  | c1
-------+----
 (0,2) |  2
(1 row)

-- Run the VACUUM command and then insert data to reuse the old space.
gaussdb=# VACUUM ANALYZE tbl_test;
gaussdb=# INSERT INTO tbl_test VALUES (3);
gaussdb=# SELECT ctid,* FROM tbl_test;
 ctid  | c1
-------+----
 (0,1) |  3
 (0,2) |  2
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
```

- **VACUUM FULL**

  - Examples

```
-- Create a table.
gaussdb=# CREATE TABLE tbl_test2(c1 int);

-- Insert 100,000 data records and check the table size.
gaussdb=# INSERT INTO tbl_test2 VALUES (generate_series(1,100000));
gaussdb=# SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 3048 kB
(1 row)

-- Delete data and check the table size.
gaussdb=# DELETE FROM tbl_test2;
gaussdb=# SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 3048 kB
(1 row)

-- Use VACUUM FULL to reclaim space and check the table size.
gaussdb=# VACUUM FULL ANALYZE tbl_test2;
gaussdb=# SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 0 bytes
(1 row)
```

2. By triggering mode, they are divided into manual VACUUM and automatic cleanup (autovacuum), as introduced above. Manual VACUUM is actively executed by users, offering high controllability. On the other hand, autovacuum is automatically triggered by a background daemon based on dead tuple thresholds, making it ideal for routine maintenance. It is worth noting that concurrent service DDL statements can terminate an autovacuum process, but they cannot interrupt a manually executed VACUUM.

   - **Automatic cleanup (autovacuum)**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 8-3 autovacuum parameters

Parameter	Default Value	Description
autovacuum	on	Specifies whether autovacuum is enabled.
autovacuum_naption	600	Specifies the interval between two autovacuum operations. The default value is 600 seconds.
autovacuum_mode	mix	Controls the operation of autovacuum threads. The options are as follows: • **analyze**: Only autoanalyze is executed. • **vacuum**: Only autovacuum is executed. • **mix**: Both autoanalyze and autovacuum are executed. • **none**: Neither autoanalyze nor autovacuum is executed.
autovacuum_max_workers	3	Maximum number of autovacuum worker threads that can run at the same time.
autovacuum_vacuum_scale_factor	0.2	Determines whether to trigger VACUUM. Value range: 0.0–100.0.
autovacuum_vacuum_threshold	50	Determines whether to trigger VACUUM. Value range: 0–2147483647.
autovacuum_analyze_scale_factor	0.1	Determines whether to trigger ANALYZE. Value range: 0.0–100.
autovacuum_analyze_threshold	50	Determines whether to trigger ANALYZE. Value range: 0–2147483647.
autovacuum_vacuum_cost_limit	**-1**, indicating that the parameter is the same as **vacuum_cost_limit** (a parameter that controls manual VACUUM).	Cost value used by the autovacuum operation. Autovacuum does not enter the sleep state until the cost reaches the value.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Default Value	Description
autovacuum_vacuum_cost_delay	**-1**, indicating that the parameter is the same as **vacuum_cost_delay** (a parameter that controls manual VACUUM).	Specifies the sleep time after the autovacuum operation's cost reaches **autovacuum_vacuum_cost_limit**.

When the number of dead tuples reaches the threshold, autovacuum is triggered. The formula for calculating the threshold is as follows:

```
number_of_dead_tuples > autovacuum_vacuum_scale_factor * reltuples + autovacuum_vacuum_threshold
```

3. By target object, VACUUM operations can be applied to a wide variety of database entities, including ordinary tables, system catalogs, TOAST tables, partitioned tables, indexes, and compressed tables (ILM). The support matrix for VACUUM, autovacuum, and VACUUM FULL across these different objects is detailed in the table below.

Table 8-4 VACUUM objects

Object	VACUUM	AUTOVACUUM	VACUUM FULL
Ordinary table	Yes	Yes	Yes
System catalog	Yes	Yes	Yes
TOAST table	Yes	Yes	Yes
EnToast table	Yes	Yes	No
Partitioned Tables	Yes	Yes	Yes
Subpartitioned table	Yes	Yes	Yes
Temporary table	Yes	No	Yes
Global temporary table	Yes	No	Yes
Index	Yes	Yes	No
Compressed table (with a specified ILM policy)	Yes	Yes	Yes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Differences Between VACUUM and VACUUM FULL

Both VACUUM and VACUUM FULL are core commands used in GaussDB to clean up tablespaces and handle dead tuples. Their common purpose is to reclaim the physical space occupied by obsolete data and mitigate table bloat.

The core difference between them lies in their cleanup mechanisms and their operational impact on active services. Standard VACUUM merely marks the space of dead tuples as reusable without physically deleting them or returning the space to the OS. During this process, it acquires only a level-4 lock (ShareUpdateExclusiveLock), which does not interfere with normal read and write workloads. VACUUM FULL completely rebuilds the original table, physically purging dead tuples and releasing the reclaimed space back to the OS. However, during its execution, it acquires a level-8 lock (AccessExclusiveLock), which blocks all read and write access from other transactions.

This fundamental difference dictates their distinct deployment scenarios: VACUUM serves as a lightweight, non-blocking routine maintenance tool that is ideal for frequent daily execution; conversely, VACUUM FULL is a heavyweight operation whose execution time scales with data volume, making it best reserved for thoroughly reclaiming space from severely bloated tables during off-peak maintenance windows.

Comparison Dimension	VACUUM	VACUUM FULL
Similarity	Clear dead tuples and mitigate table bloat.	
Clearance method	Mark the space occupied by dead tuples as reusable but not physically delete the tuples.	Rebuild the original table and physically delete dead tuples.
Space return	Reuse space within the database only, without returning it to the OS.	Release and return physical space to the OS.
Table locking	Obtain a level-4 lock (ShareUpdateExclusiveLock), which does not affect read/write operations.	Obtain a level-8 lock (AccessExclusiveLock), which affects read/write operations.
Impact on performance	Lightweight, with little impact.	Heavyweight, with great impact.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Comparison Dimension	VACUUM	VACUUM FULL
Application scenario	Routine cleanup.	Thorough space reclamation during off-peak hours.

Differences Between Astore VACUUM and Ustore VACUUM

This section compares the differences in VACUUM mechanisms and functionalities between GaussDB's next-generation storage engine, Ustore, and its classic counterpart, Astore. The following three tables detail the support matrix for autovacuum, VACUUM, and VACUUM FULL operations across different storage components. These include heap tables and B-tree indexes in Astore, as well as heap tables and UB-tree indexes (available in both RCR and PCR formats) in Ustore.

Table 8-5 Table 3 AUTOVACUUM

AUTOVACUUM	Astore	B-tree	Ustore	UB-tree (RCR)	UB-tree (PCR)
Scanning and deleting invalid row versions	Yes	Yes	No	Yes	Yes
Maintaining statistics	Yes	Yes	No	No	No
Maintaining FSM accuracy	Yes	Yes	Yes	Yes	Yes
Maintaining VM accuracy	Yes	No	No	No	No
Freezing outdated transaction IDs	Yes	No	No	Yes	No
Recycling transaction files	Yes	Yes	Yes	Yes	Yes
Releasing physical storage space	No	No	No	No	No

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 8-6 Table 4 VACUUM

VACUUM	Astore	B-tree	Ustore	UB-tree (RCR)	UB-tree (PCR)
Scanning and deleting invalid row versions	Yes	Yes	Yes	Yes	Yes
Maintaining statistics	Yes	Yes	Yes	Yes	Yes
Maintaining FSM accuracy	Yes	Yes	Yes	Yes	Yes
Maintaining VM accuracy	Yes	No	No	No	No
Freezing outdated transaction IDs	Yes	No	Yes	Yes	Yes
Recycling transaction files	Yes	Yes	Yes	Yes	Yes
Releasing physical storage space	No	No	No	No	No

Table 8-7 Table 5 VACUUM FULL

VACUUM FULL	Astore	B-tree	Ustore	UB-tree (RCR)	UB-tree (PCR)
Releasing physical storage space	Yes	Yes	Yes	Yes	Yes

VACUUM-Specific Features and Operations

After learning the core concepts and classifications of VACUUM, you can refer to the following sections to explore more details and configure the feature for your actual production workloads:

- VACUUM on a table: Perform cleanup operations on a specified table to reclaim storage space. For details, see "Importing Data > Executing VACUUM to a Table" in *Administrator Guide*.
- VACUUM syntax: Explore the complete syntax and parameters of the VACUUM and VACUUM FULL commands. For details, see "SQL Reference > SQL Syntax > V > VACUUM" in *Reference*.
- Automatic cleanup and parameter configuration: Control the enabling and triggering conditions of automatic cleanup using parameters such as

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

autovacuum, **autovacuum_vacuum_scale_factor**, and **autovacuum_vacuum_threshold**. For details, see "Database GUC Parameters > GUC Parameters > Autovacuum" in *Reference*, as well as "VACUUM/Autovacuum Introduction and Optimization > Autovacuum Parameters" in *Performance Optimization*.

- VACUUM/autovacuum optimization: Enhance cleanup efficiency and table bloat control. For details, see "VACUUM/Autovacuum Introduction and Optimization" in *Performance Optimization*.

8.15 TOAST Table

Description

The oversized-attribute storage technique (TOAST) is a special storage technique. Only some data in the variable-length form support TOAST, such as the text type. TOAST is used to overcome the limitation that a single field in the database is too large to be directly stored in a standard data page. In GaussDB storage mode, all columns in a record are called a tuple, and multiple tuples form a page. The basic unit of data in file storage is page. GaussDB uses a fixed page size, typically 8 KB. When the size of a row of data exceeds the page size, the database employs TOAST to ensure that these oversized fields can be effectively stored and managed.

Out-of-line storage is a part of TOAST. This technology compresses or slices large field values into multiple physical rows and stores them in another table (known as the TOAST table). The main table retains only references (usually pointers) to these large field values. This avoids performance problems caused by large data in a single row and ensures effective access to large field data.

If a table has a joined TOAST table, the **reltoastrelid** column in the PG_CLASS system catalog records the OID of the TOAST table. If the table has no such a joined TOAST table, the value of **reltoastrelid** is **0**.

GaussDB supports four storage policies for table columns:

1. PLAIN:
   - Compression and out-of-line storage are not allowed. This policy is usually used for data types that cannot use the TOAST mechanism.
   - It is applicable to data types that can be stored without TOAST, such as integer and character types.
   - This policy is not applicable to types whose storage length exceeds the page size, for example, text.
2. EXTENDED:
   - Compression and out-of-line storage are allowed. This is the default storage policy for most data types that can use the TOAST mechanism.
   - The database first attempts to compress the data. If the compressed data is still too large, the data is stored out of the line.
   - EXTENDED is the default policy for most TOAST data types, such as text and bytea.
3. EXTERNAL:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Out-of-line storage is allowed, but compression is not allowed.
  - This strategy makes string operations of the text and bytea data types faster because the entire row of data does not need to be read and decompressed.

4. MAIN:
  - Compression is allowed.
  - It is mainly applicable to data of the numeric type.
  - The database first attempts to compress the data. If the compressed data is still too large, the data is stored out of the line.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE tb_t1(id int, info text);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
gaussdb=# CREATE TABLE tb_t2(id int, age int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

-- Check the information of the tb_t1 table. The info column is of the text type and Storage is extended.
-- Below is part of the command output:
gaussdb=# \d+ tb_t1
                        Table "public.tb_t1"
 Column |  Type   | Modifiers | Storage  | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
 id     | integer |           | plain    |              |
 info   | text    |           | extended |              |


-- Check the information of the tb_t2 table. All columns are of the int type.
-- Below is part of the command output:
gaussdb=# \d+ tb_t2
                            Table "public.tb_t2"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 age    | integer               |           | plain    |              |

-- The following query shows that the tb_t1 table has joined TOAST tables, while the tb_t2 table does not have any joined TOAST tables.
gaussdb=# SELECT oid,relname,reltoastrelid FROM pg_class WHERE relname in ('tb_t1','tb_t2');
  oid  | relname | reltoastrelid
-------+---------+---------------
 34519 | tb_t1   |         34522
 34524 | tb_t2   |             0
(2 rows)

-- Query information about the TOAST table joined with tb_t1.
gaussdb=# SELECT relname,oid FROM pg_class WHERE oid = '34522';
    relname     |  oid
----------------+-------
 pg_toast_34519 | 34522
(1 row)

-- Drop.
gaussdb=# DROP TABLE tb_t1,tb_t2;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.16 HTAP

Description

The hybrid transactional/analytical processing (HTAP) database architecture is a technical innovation solution that revolutionizes the traditional separation of transactional workload (OLTP) and analytical workload (OLAP) systems. The architecture uses a unified data storage engine and compute resource scheduling mechanism to implement collaborative operation of OLTP and OLAP in the same architecture, effectively eliminating the problem of low analysis efficiency caused by data redundancy, delayed synchronization, and resource isolation in the traditional architecture, and providing a unified data infrastructure to support real-time decision-making.

Based on the traditional row-store engine and execution engine, GaussDB's HTAP architecture innovatively integrates the in-memory column view (IMCV), transparent routing mechanism for hybrid loads, and vectorized engine to build a complete hybrid row-column store system. The architecture uses SQL-level data migration APIs to convert row-store data into IMCV in real time, and uses the memory persistence technology to ensure strong consistency of TP/AP operations, effectively solving the problem of delayed data synchronization in the traditional hybrid row-column store architecture. The core breakthroughs include: The intelligent routing engine based on the multidimensional cost model can parse hybrid load characteristics in real time, automatically direct OLTP requests to the row-store engine to ensure transaction performance, and accurately route complex analysis queries to the IMCV engine to effectively improve performance. Finally, the unified SQL APIs provide real-time non-aware analysis capabilities.

NOTE

Currently, HTAP supports only Ustore.

Examples

This section uses a complete execution process as an example to describe how to use HTAP for hybrid load queries.

Step 1 Enable HTAP and set related parameters. For details, see "Best Practices for HTAP > Preparations" in *Best Practices*.

In distributed mode, HTAP supports only ORA-compatible mode. Therefore, you need to create an ORA-compatible database and perform all subsequent operations in the database.

```
htapdb=# CREATE DATABASE htapdb WITH DBCOMPATIBILITY='ORA';
CREATE DATABASE
htapdb=# \c htapdb
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "htapdb" as user "xxx".
```

Step 2 Execute HTAP DDL statements to create IMCV metadata and load row-store data to IMCV.

- Create an IMCV for an existing table using ALTER.

  ```
  -- Scenario 1:
  -- Create an IMCV for all columns in an existing ordinary table.
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
htapdb=# DROP TABLE IF EXISTS htap_test;
NOTICE:  table "htap_test" does not exist, skipping
DROP TABLE
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20), comments
varchar(20)) WITH (storage_type = ustore);
CREATE TABLE
htapdb=# INSERT INTO htap_test VALUES (generate_series(1,3000), 1, 1000, 'large', 'large table');
INSERT 0 3000
htapdb=# ALTER TABLE htap_test COLVIEW;
ALTER TABLE

-- Scenario 2:
-- Create an IMCV for specified columns in an existing ordinary table.
htapdb=# DROP TABLE IF EXISTS htap_test;
DROP TABLE
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20), comments
varchar(20)) WITH (storage_type = ustore);
CREATE TABLE
htapdb=# INSERT INTO htap_test VALUES (generate_series(1,3000), 1, 1000, 'large', 'large table');
INSERT 0 3000
htapdb=# ALTER TABLE htap_test COLVIEW(id, dept_id, salary);
ALTER TABLE

-- Scenario 3:
-- Create an IMCV for specified partitions in an existing partitioned table.
htapdb=# DROP TABLE IF EXISTS htap_test;
DROP TABLE
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20), comments
varchar(20)) WITH (storage_type = ustore)
PARTITION BY RANGE(id)
(
PARTITION p1 VALUES LESS THAN (1001),
PARTITION p2 VALUES LESS THAN (2001),
PARTITION p3 VALUES LESS THAN (3001)
);
CREATE TABLE
htapdb=# INSERT INTO htap_test VALUES (generate_series(1,3000), 1, 1000, 'large', 'large table');
INSERT 0 3000
-- Create an IMCV only for partition p1.
htapdb=# ALTER TABLE htap_test MODIFY PARTITION p1 COLVIEW;
ALTER TABLE
```

- Create an IMCV when creating a table using CREATE.

```
-- Scenario 1:
-- Create an IMCV for all columns when creating a table.
htapdb=# DROP TABLE IF EXISTS htap_test;
DROP TABLE
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20), comments
varchar(20)) COLVIEW WITH (storage_type = ustore);
CREATE TABLE

-- Scenario 2:
-- Create an IMCV for specified columns when creating a table.
htapdb=# DROP TABLE IF EXISTS htap_test;
DROP TABLE
-- Specify the columns (id, dept_id, name) to be loaded.
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20) NOCOLVIEW,
comments varchar(20) NOCOLVIEW) COLVIEW WITH (storage_type = ustore);
CREATE TABLE

-- Scenario 3:
-- Create an IMCV for specified partitions when creating a table.
htapdb=# DROP TABLE IF EXISTS htap_test;
DROP TABLE
htapdb=# CREATE TABLE htap_test(id int, dept_id int, salary int, name varchar(20), comments
varchar(20)) COLVIEW WITH (storage_type = ustore)
PARTITION BY RANGE(id)
(
PARTITION p1 VALUES LESS THAN (1001),
PARTITION p2 VALUES LESS THAN (2001) NOCOLVIEW,
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PARTITION p3 VALUES LESS THAN (3001) NOCOLVIEW
    );
    CREATE TABLE
```

Step 3 View the basic information about the table where an IMCV is created.

```
htapdb=# SELECT reloid,relname,imcvnattr,imcvkey FROM gs_imcv;
 reloid |  relname  | imcvnattr | imcvkey
--------+-----------+-----------+---------
  33248 | htap_test |         3 | 1 2 3
(1 row)
-- Use the gs_htap_imcv_info() function to query the basic information about the table, including the
number of loaded row groups and the amount of column-store data.
htapdb=# SELECT * FROM gs_htap_imcv_info();
 rel_name  | partition_name | subpartition_name | schema_name | user_name | imcvnattr | imcvkey | priority
| rowgroup_num | populated_rowgroup_num | cu_num_in_
mem | cu_mem_size | cu_num_in_disk | cu_disk_size
-----------+----------------+-------------------+-------------+-----------+-----------+---------+----------+--------------
+------------------------+-----------
----+-------------+----------------+--------------
 htap_test |                |                   | public      | ****      |         3 | 1 2 3   |        1 |            0 |                        0
|
  0 |           0 |             0 |            0
(1 row)
```

Step 4 Execute a query and accurately route the query to the row-store or column-store engine based on the service load characteristics.

```
-- Automatic transparent routing mode: A row-store, column-store, or hybrid execution plan is selected
automatically based on whether column-store is loaded for related tables and columns and based on the
row-store/column-store plan costs.
htapdb=# SET htap_router_mode='auto';
SET
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM htap_test; -- The column-store plan is selected
due to its lower cost.
              QUERY PLAN
---------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on htap_test
(4 rows)
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM htap_test; -- The loaded columns do not meet the query
requirements. A row-store plan is selected.
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on htap_test
(3 rows)

-- Forcible column store mode: If the queried tables and columns are accelerated, a column-store plan is
automatically executed. If no such acceleration is performed or the loaded columns do not meet the query
requirements, a row-store plan will be executed.
htapdb=# SET htap_router_mode='column';
SET
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM htap_test; -- Forcibly execute a column-store
plan.
              QUERY PLAN
---------------------------------------
 Row Adapter
   ->  Vector Streaming (type: GATHER)
         Node/s: All datanodes
         ->  Imcv Scan on htap_test
(4 rows)
htapdb=# EXPLAIN (COSTS OFF) SELECT * FROM htap_test; -- The loaded columns do not meet the query
requirements. A row-store plan is executed.
         QUERY PLAN
-----------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  ->  Seq Scan on htap_test
(3 rows)

-- Forcible row store mode: A row-store plan is executed for querying loads regardless of whether HTAP
column-store is loaded.
htapdb=# SET htap_router_mode='row';
SET
htapdb=# EXPLAIN (COSTS OFF) SELECT id,dept_id FROM htap_test; -- Forcibly execute a row-store plan.
        QUERY PLAN
------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on htap_test
(3 rows)
```

Step 5 (Optional) Unload the IMCV.

```
-- Unload the IMCV table. After the IMCV table is unloaded, you need to perform Step 2 again to use HTAP.
htapdb=# ALTER TABLE htap_test NOCOLVIEW;
ALTER TABLE
htapdb=# SELECT * FROM gs_htap_imcv_info();
 rel_name | partition_name | subpartition_name | schema_name | user_name | imcvnattr | imcvkey | priority |
rowgroup_num | populated_rowgroup_num | cu_num_in_mem |
cu_mem_size | cu_num_in_disk | cu_disk_size
----------+----------------+-------------------+-------------+-----------+-----------+---------+----------+--------------
+------------------------+---------------+-
------------+----------------+--------------
(0 rows)
```

----End

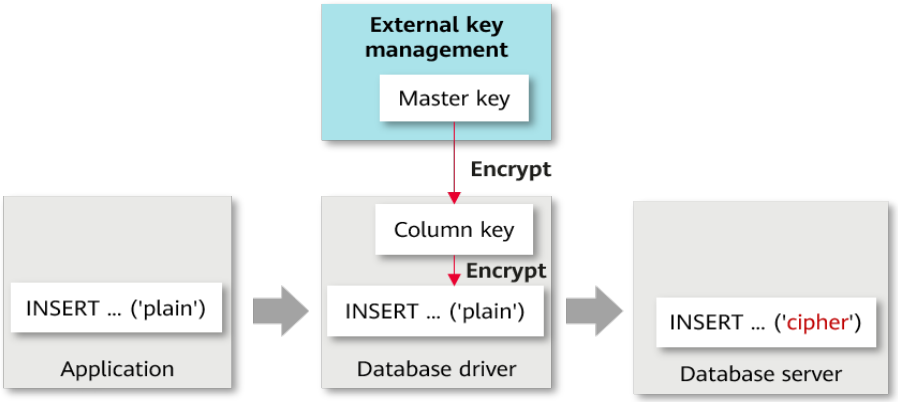
8.17 Fully-Encrypted Database

A fully-encrypted database can protect privacy throughout the data lifecycle. Data is always encrypted during transmission, computing, and storage, regardless of the service scenarios or environments, and users maintain control over their encryption keys.

Encryption Model

A fully-encrypted database uses a multi-level encryption model comprising three core objects: data, column key, and master key. Their relationships are illustrated in **Figure 8-1**.

Figure 8-1 Encryption model



Below is an introduction to the three objects:

- **Data**, including:
  a. Data contained in SQL syntax, such as 'data' in INSERT... VALUES ('data')
  b. Query results returned from the database server, such as the results of the SELECT syntax

  NOTE

  An encrypted database encrypts data of encrypted columns in the SQL syntax in the driver and decrypts the query result of the encrypted columns returned from the database server.
- **Column key**: Data is encrypted by using column keys. The column keys are generated by the database driver or manually imported by users. The column key ciphertext is stored on the database server.
- **Master key**: Column keys are encrypted by using master keys. The master keys are generated and stored by an external key manager. The database driver automatically accesses the external key manager to encrypt and decrypt column keys.

Key Management

Currently, the fully-encrypted database supports the following tools for independent key management:

- Huawei KMS: an online key management service provided by Huawei Cloud. It provides functions such as creating, deleting, querying, and backing up keys, and supports online data encryption and decryption using keys.
- user_token: The key is derived from the password provided by users on the client.
- sdf_kms: supports intrinsic key management based on the Kunpeng TEE and the SanSec SJJ1994 HSM.

Database Driver

Currently, fully-encrypted databases support the gsql, JDBC, and Go drivers.

Using gsql to Execute SQL Statements

Generate master keys and confirm their access parameters before executing the SQL statements in this section.

This section uses a complete execution process as an example to describe how to use the encrypted database syntax, including three phases: DDL statement execution, DML statement execution, and cleanup.

Step 1 Connect to the database and use the **-C** parameter to enable the full encryption function.

```
gsql -p PORT -d DATABASE -h HOST -U USER -W PASSWORD -r -C
```

Step 2 Set parameters for accessing the master key using a meta-command.

Note: There must be no line feed or space in the string starting from **keyType**. Otherwise, the gsql tool cannot identify the entire parameter.

Key management based on the SanSec SJJ1994 HSM:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# \key_info keyType=sdf_kms,library_path={Path of the dynamic library},library_version=SJJ1994,kek_algo=SGD_SM4_CBC,mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC
```

NOTE

- Before using the SanSec SJJ1994 HSM, you must correctly configure the .ini file and ensure that the HSM can be properly connected.
- **library_path** can be an absolute path or a relative path (relative to *$GAUSSHOME*/**lib**).
- **kek_algo** refers to the algorithm that key encryption keys (KEKs) use to encrypt session keys. Supported algorithms include: SGD_SM4_ECB (default), SGD_SM4_CBC, SGD_SM4_CTR, SGD_AES_ECB, SGD_AES_CBC, and SGD_AES_CTR.
- **dkey_algo** refers to the algorithm that session keys use to encrypt data keys. Supported algorithms include: SGD_SM4_CBC, SGD_SM4_CTR, SGD_AES_CBC, and SGD_AES_CTR.
- **mac_algo** refers to the algorithm that session keys use to verify the integrity of data keys. Supported algorithms include: SGD_SM4_MAC and SGD_AES_MAC.
- **library_version** (optional) is used for customized interconnection. Currently, it can only be set to **SJJ1994**.
- **kek_algo** must match the **KEY_PATH** specified when the CMK is created. That is, both of them must be AES or SM4 series algorithms.
- Currently, the gsql and JDBC drivers are supported.

Kunpeng TEE intrinsic key management:

```
gaussdb=# \key_info keyType=sdf_kms,library_path={Path of the dynamic library},mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC
```

Example: \key_info KeyType=sdf_kms,library_path=./libsdf.so,mac_algo=SGD_SM4_MAC,dkey_algo=SGD_SM4_CBC

Huawei Cloud offers two authentication modes. These modes differ in the number and types of parameters required. You can choose either mode as required.

- **Authentication mode 1**: AK/SK authentication

  ```
  gaussdb=# \key_info keyType=huawei_kms,kmsProjectId={Project ID},ak={AK},sk={SK}
  ```

  Parameters: For details about how to obtain parameters, including the project ID, AK, and SK, see the master key generation phase.

  Example: \key_info keyType=huawei_kms,kmsProjectId=0b59929e8100268a2f22c01429802728,ak=XMAUMJY******DFWLQW,sk=ga6rO8lx1Q4uB*********2gf80muIzUX

- **Authentication mode 2**: Account/Password authentication

  ```
  gaussdb=# \key_info keyType=huawei_kms,iamUrl={IAM server address},iamUser={IAM username},iamPassword={IAM user password},iamDomain={Account name},kmsProject={Project}
  ```

  Parameters: For details about how to obtain related parameters, including the IAM server address, IAM username, IAM user password, account name, and project, see the master key generation phase.

  Example: \key_info keyType=huawei_kms,iamUrl=https://iam.*example*.com/v3/auth/tokens,iamUser=test,iamPassword=*********,iamDomain=test_account,kmsProject=xxx

Step 3 Define master keys.

In the master key generation phase, the KMS has generated and stored master keys. Running this syntax only stores the master key information into the database for future access. For details about the syntax format, see "SQL Reference > SQL Syntax > C > CREATE CLIENT MASTER KEY" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# CREATE CLIENT MASTER KEY cmk2 WITH ( KEY_STORE = sdf_kms, KEY_PATH = '{KEK ID}');
gaussdb=# CREATE CLIENT MASTER KEY cmk1 WITH ( KEY_STORE = huawei_kms , KEY_PATH = '{KMS server address}|{Key ID}', ALGORITHM = AES_256);
CREATE CLIENT MASTER KEY
```

- Parameters: For details about how to obtain related parameters, including the KMS server address and key ID, see the master key generation phase.

  Example of *KEY_PATH*: https://kms.cn-north-4.myhuaweicloud.com/v1.0/0b59929e8100268a2f22c01429802728/kms/9a262917-8b31-41af-a1e0-a53235f32de9

Step 4 Define column keys.

Column keys are encrypted by the master keys defined in the previous step. For details about the syntax, see "SQL Reference > SQL Syntax > C > CREATE COLUMN ENCRYPTION KEY" in *Reference*.

```
gaussdb=# CREATE COLUMN ENCRYPTION KEY cek1 WITH VALUES (CLIENT_MASTER_KEY = cmk1, ALGORITHM  = AES_256_GCM);
gaussdb=# CREATE COLUMN ENCRYPTION KEY cek2 WITH VALUES (CLIENT_MASTER_KEY = cmk2, ALGORITHM = SM4_HMAC_SM3);
```

Step 5 Define an encrypted table.

In this example, the **name** and **credit_card** columns in the table are specified as encrypted columns by using syntax.

```
gaussdb=# CREATE TABLE creditcard_info (
  id_number int,
  name text encrypted with (column_encryption_key = cek1, encryption_type = DETERMINISTIC),
  credit_card varchar(19) encrypted with (column_encryption_key = cek1, encryption_type = DETERMINISTIC));
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'id_number' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

Step 6 Perform other operations on the encrypted table.

```
-- Write data to the encrypted table.
gaussdb=# INSERT INTO creditcard_info VALUES (1,'joe','6217986500001288393');
INSERT 0 1
gaussdb=# INSERT INTO creditcard_info VALUES (2, 'joy','6219985678349800033');
INSERT 0 1

-- Query data from the encrypted table.
gaussdb=# select * from creditcard_info where name = 'joe';
 id_number | name |     credit_card
-----------+------+---------------------
         1 | joe  | 6217986500001288393

-- Update data in the encrypted table.
gaussdb=# update creditcard_info set credit_card = '80000000011111111' where name = 'joy';
UPDATE 1

-- Add an encrypted column to the table.
gaussdb=# ALTER TABLE creditcard_info ADD COLUMN age int ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = cek1, ENCRYPTION_TYPE = DETERMINISTIC);
ALTER TABLE

-- Drop an encrypted column from the table.
gaussdb=# ALTER TABLE creditcard_info DROP COLUMN age;
ALTER TABLE

-- Query master key information from the system catalog.
gaussdb=# SELECT * FROM gs_client_global_keys;
 global_key_name | key_namespace | key_owner | key_acl |        create_date
-----------------+---------------+-----------+---------+----------------------------
 cmk1            |          2200 |        10 |         | 2021-04-21 11:04:00.656617
(1 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Query column key information from the system catalog.
gaussdb=# SELECT column_key_name,column_key_distributed_id ,global_key_id,key_owner  FROM
gs_column_keys;
 column_key_name | column_key_distributed_id | global_key_id | key_owner
-----------------+---------------------------+---------------+-----------
 cek1            |                760411027 |         16392 |        10
(1 rows)

-- View the meta information of columns in the table.
gaussdb=# \d creditcard_info
        Table "public.creditcard_info"
  Column    |       Type        | Modifiers
------------+-------------------+------------
 id_number  | integer           |
 name       | text              |  encrypted
 credit_card | character varying |  encrypted
```

Step 7 Clean up.

```
-- Drop the encrypted table.
gaussdb=# DROP TABLE creditcard_info;
DROP TABLE

-- Drop the column keys.
gaussdb=# DROP COLUMN ENCRYPTION KEY cek1;
DROP COLUMN ENCRYPTION KEY
gaussdb=# DROP COLUMN ENCRYPTION KEY cek2;
DROP COLUMN ENCRYPTION KEY

-- Drop the master keys.
gaussdb=# DROP CLIENT MASTER KEY cmk1;
DROP CLIENT MASTER KEY
gaussdb=# DROP CLIENT MASTER KEY cmk2;
DROP CLIENT MASTER KEY
```

----End

8.18 DDL

Description

This feature changes database objects. GaussDB supports the following DDL functions:

- Index operations
- Primary key operations
- Column operations
- Generated column operations
- Foreign key operations
- Table operations
- Partition operations
- Schema operations
- Database instance operations
- Tablespace operations

Based on whether the target object data of DDL operations is processed, DDL statements are classified into the following types:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Logical DDL statements: modify only the metadata of database objects, and do not process the data stored in the objects. For example, change the table name or column name.
- Physical DDL statements: modify both the metadata of the changed objects and the user data stored in the objects. For example, to create an index for a table, you need to modify the table definition and scan the entire table to build a new index.

Principles

- Logical DDL statements: Only the metadata of database objects needs to be modified. Therefore, the execution time is usually short, only dozens of milliseconds or several seconds. Executing these DDL statements consumes almost no system resources and therefore does not affect service loads.
- Physical DDL statements: The execution time is long, and is affected by the data volume in the table, machine configuration, and service loads. An exclusive lock is added during offline DDL operations. The lock is released after data reorganization and metadata modification are complete. Therefore, service execution is blocked during the process. Physical DDL statements block services for a long time. Therefore, GaussDB provides online DDL operations to reduce the impact on services and improve service continuity and database availability. No exclusive lock is added for online DDL operations. Concurrent read and write operations can be performed on the table during online DDL operations, reducing the impact on user services.

Examples

This section describes the common basic DDL functions of GaussDB, such as column and table operations.

Step 1 Create a table.

```
gaussdb=# CREATE TABLE t1(c1 INT,c2 INT);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,10), generate_series(1,10));
INSERT 0 10
```

Step 2 Adds a column.

```
gaussdb=# ALTER TABLE t1 ADD COLUMN c3 INTEGER;
ALTER TABLE
```

Step 3 Drop a column.

```
gaussdb=# ALTER TABLE t1 DROP COLUMN c3;
ALTER TABLE
```

Step 4 Rename a table.

```
gaussdb=# ALTER TABLE t1 RENAME TO t2;
ALTER TABLE
gaussdb=# ALTER TABLE t2 RENAME TO t1;
ALTER TABLE
```

Step 5 Change column types.

- Change the column type online.

  ```
  gaussdb=# ALTER TABLE ONLINE t1 ALTER c2 TYPE BIGINT;
  ALTER TABLE
  ```

- Change the column type offline.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# ALTER TABLE t1 ALTER c2 TYPE INT;
ALTER TABLE
```

----End

See Also

- For offline DDL operations, see "SQL Reference > SQL Syntax > DDL Syntax Overview" in *Reference*.
- For online DDL operations, see **Online DDL Operations** and "SQL Reference > SQL Syntax > Online DDL and Related Operations" in *Reference*.

8.19 Write Ahead Logs

Write Ahead Log (WAL for short, also referred to as Xlog) is a technology used to ensure data reliability and system recoverability. The core idea is that before modifying a data file, you must record the modification operations in a log file and flush the log file to the persistent storage. You can use WALs to restore a cluster if the system crashes. WALs can be recorded at different levels, which are specified by the GUC parameter **wal_level**.

Principles

- Log Running Mechanism

  When GaussDB is started, a memory segment is allocated to cache logs generated during database running. The buffer size is specified by the **wal_buffers** parameter. To ensure transaction persistence, logs in **wal_buffers** must be flushed to disks.

  WALs are written to disks in the following scenarios:

  - Commit a transaction using COMMIT. The system calls the fdatasync function to write WAL data in **wal_buffers** to WAL files. (**wal_sync_method** can control the function called during commit. The function fdatasync is called by default.)
  - WAL writer periodically writes data. The WAL writer process periodically writes WAL data in **wal_buffers** to WAL files. The **wal_writer_delay** parameter specifies the interval at which the WAL writer process works. The default value is **200**, in ms. A longer delay might lead to insufficient WAL buffer and a shorter delay leads to continuously writing of the WALs, thereby increasing the load of disk I/O.

- Log Storage Directory

  The path is *Data directory*/**pg_xlog**.

  - The WAL storage path of CNs is /*<CN data directory>*/**pg_xlog**. The default size of a WAL file is 16 MB, which is specified by the **wal_segment_size** parameter. Generally, you do not need to change the size.

    ```
    gaussdb=# SHOW wal_segment_size;
    wal_segment_size
    ------------------
    16MB
    (1 row)
    ```

  - The following uses one DN as an example. You can run the following command to query the data directory: **/opt/gaussdb/data/dn/db_6001**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

indicates the data directory of the cluster node. Xlogs on the current node are stored in the **/opt/gaussdb/data/dn/db_6001/pg_xlog** directory.

```
$ cm_ctl query -Cvd
[   CMServer State   ]

node              instance                       state
------------------------------------------------------------
1  192.168.10.182 1    /opt/gaussdb/data/cm/cm_server Primary

[    Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[   Datanode State   ]

node              instance                       state
------------------------------------------------------------
1  192.168.10.182 6001 /opt/gaussdb/data/cm/cm_server Primary
```

- Log Naming Rules

  WALs are saved as segment files. Each segment is 16 MB and is divided into multiple 8 KB pages. The name of a segment file consists of 24 hexadecimal characters. The name of a WAL file consists of the timeline, log file identifier, and log file segment identifier. For details, see **Table 8-8**.

  The upper limit of the segment identifier of a WAL file is **FF**. If there is another WAL file, add 1 to the file identifier and reset the segment identifier to **0**. For example, the next file of 000000010000000000000FF is 000000010000000100000000. The timeline starts from 1, and the log file identifier and the segment identifier of the file both start from 0.

  Table 8-8 WAL file naming rules

Column	Description
Timeline	Time line The value is an 8-digit hexadecimal number.
wal_file_num	Identifier of a WAL file The value is an 8-digit hexadecimal number.
wal_segment_num	Identifier of a WAL file segment The value is an 8-digit hexadecimal number.

  The size of each WAL file is 16 MB. Typical WAL file names are as follows:

```
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 11  12:33 000000010000000100000F2
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 11  18:10 000000010000000100000F3
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 11  22:18 000000010000000100000F4
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 12 03:58 000000010000000100000F5
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 12 07:59 000000010000000100000F6
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 12 13:45 000000010000000100000F7
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 12 17:30 000000010000000100000F8
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 12 23:10 000000010000000100000F9
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 13 03:10 000000010000000100000FA
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 13 08:39 000000010000000100000FB
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 13 12:43 000000010000000100000FC
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 13 17:08 000000010000000100000FD
-rW----- 1 gaussdb d gaussdb d 16777216 Jun 13 16:32 000000010000000100000FE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> The numbers in the WAL file name are used in ascending order in succession. Exhausting all available numbers takes a long time, and the numbers will start from zero again after they reach the maximum.

- Log Content Description

  The content of WALs depends on the types of recorded transactions. WALs can be used to restore a system after the system breaks down. By default, GaussDB reads WALs for system restoration during each startup.

- Log Application Scenario
  - Database recovery: When a database crashes unexpectedly, database changes have been recorded in Xlogs. After the database is restarted, the Xlogs after the checkpoint can be replayed to flush the modified data pages (recorded in the Xlogs but not modified in the data files) to disks, ensuring data integrity. This process is automatically completed after the database is started.
  - Data synchronization between the primary and standby DNs: The standby DN of GaussDB receives Xlogs from the primary DN and applies Xlogs to ensure data consistency between the primary and standby DNs. Data synchronization between the primary and standby databases can also be implemented through Xlogs.
  - Database backup and restoration: When GaussRoach is used to back up GaussDB, you need to copy data files and Xlogs generated during the backup to form a complete backup. You can use a full backup or an incremental backup and Xlogs to restore data to a specified point in time, that is, perform PITR.
  - Logical replication: Logical replication tools, such as DRS, parse Xlogs to obtain data streams for incremental data synchronization.

- Xlog Recycling

  During database running, Xlogs are continuously generated. GaussDB clears Xlogs using the checkpoint mechanism (also called Xlog recycling). For details about parameters related to Xlog recycling, see **Table 8-9**. For details about the parameters and how to set the parameters, see "Database GUC Parameters" in *Reference*.

Table 8-9 Parameters related to Xlog recycling

Parameter	Description
checkpoint_timeout	Sets the maximum time between automatic checkpoints. Specifically, this is the interval for the next automatic checkpoint after the last automatic checkpoint. The default value is 15 minutes. If the number of Xlogs accumulated between two automatic checkpoints is greater than the value of **checkpoint_segments**, an automatic checkpoint is triggered.
checkpoint_segments	Maximum number of WAL files that can be retained within the interval specified by **checkpoint_timeout**. The size of each log file is 16 MB.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
enable_incremental_checkpoint	Specifies whether to enable incremental checkpointing. The default value is **on**.
incremental_checkpoint_timeout	Specifies the maximum interval between automatic checkpoints when the incremental checkpoint (**enable_incremental_checkpoint**) is enabled. The default value is 1 minute. That is, an incremental checkpoint is started every minute, and whether Xlogs need to be cleared is checked.
wal_keep_segments	Specifies the minimum number of Xlog files that can be retained in the **pg_xlog** directory. The standby node obtains the Xlogs from the primary node to perform streaming replication. The default value is **128**. Generally, when executing a checkpoint, the database checks whether the number of Xlogs in the **pg_xlog** directory is greater than the value of **wal_keep_segments**. If the number is greater than the value of **wal_keep_segments**, the database deletes the earliest Xlogs until the number of Xlogs is equal to the value of **wal_keep_segments**.
enable_xlog_prune	Specifies whether the primary node recycles logs when any standby node is disconnected and the size of Xlogs exceeds the value of **max_size_for_xlog_prune**. The value **on** indicates that the functionality is enabled. When a standby node is disconnected, Xlogs between the primary and standby DNs are not cleared to ensure that Xlogs are continuous. However, if **enable_xlog_prune** is enabled and the accumulated Xlog size is greater than the value of **max_size_for_xlog_prune** (generally 256 GB), Xlogs are cleared.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description
max_size_for_xlog_prune	Maximum number of Xlogs retained on the primary node when the standby node is faulty. This parameter takes effect when **enable_xlog_prune** is enabled. The mechanism is as follows: 1. If all standby nodes specified by the replconninfo series GUC parameters are connected to the primary node, this parameter does not take effect. 2. This parameter takes effect when one or more standby nodes specified by the replconninfo series GUC parameters are disconnected from the primary node. If the number of Xlogs on the primary node is greater than the value of this parameter, the Xlogs will be forcibly recycled. Exception: In synchronous commit mode (that is, the **synchronous_commit** parameter is not set to **local** or **off**), if there are standby nodes connecting to the primary node, the primary node retains the logs after the minimum log receiving location in the majority of standby nodes. In this case, the number of retained logs may be greater than the value of **max_size_for_xlog_prune**. 3. If a standby node is being built, this parameter does not take effect. All logs on the primary node are retained to prevent failures caused by log recycling during the build operation.

Examples

- Query the size of a single WAL file.

```
gaussdb=# SHOW wal_segment_size;
wal_segment_size
-----------------
16MB
(1 row)
```

- Query the WAL data directory.

```
$ cm_ctl query -Cvd
[   CMServer State   ]

node              instance                       state
------------------------------------------------------------------
1  192.168.10.182  1    /opt/gaussdb/data/cm/cm_server Primary

[    Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[   Datanode State   ]

node              instance                      state
-------------------------------------------------------------------------
1  192.168.10.182  6001 /opt/gaussdb/data/dn/dn_6001 P Primary Normal
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.20 Ultimate RTO

Ultimate Recovery Time Objective (RTO) makes full use of multi-core CPU resources of servers and establishes multi-level log replay pipelines to improve the log replay concurrency on standby nodes to page level, achieving high availability through replay acceleration.

The primary node in GaussDB sends WALs to the standby node for replay to implement primary/standby synchronization. When the service pressure is too high, if the replay speed of the standby node cannot catch up with that of the primary node, logs accumulate on the standby node after the system runs for a long time. In this case, if the primary node is faulty, it takes a long time to restore the database, affecting system availability. If hardware resources are sufficient, enabling ultimate RTO can accelerate the replay of the standby node, shorten the database recovery time after the primary node is faulty, and improve system availability.

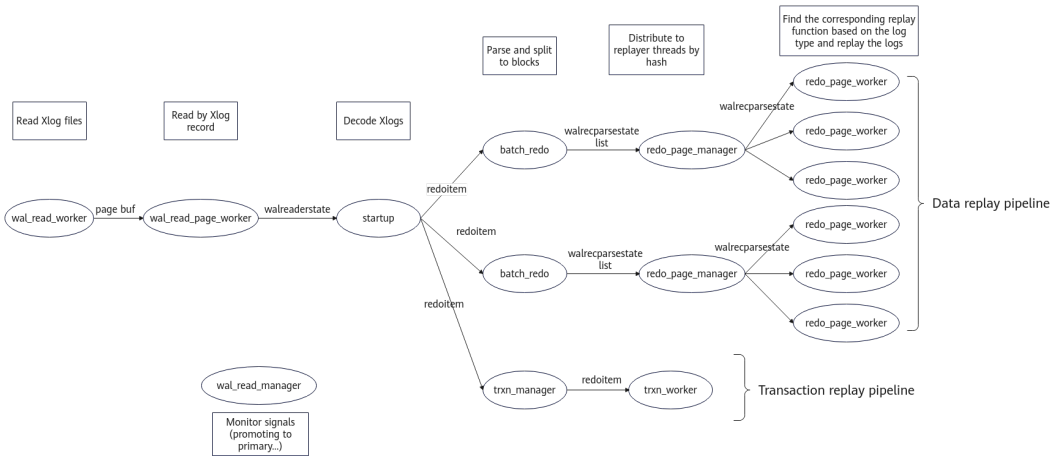
To enable ultimate RTO, set the **recovery_parse_workers** and **recovery_redo_workers** parameters. **recovery_parse_workers** indicates the number of data replay pipelines to be enabled, and **recovery_redo_workers** indicates the number of data replayer threads to be started on each data replay pipeline. If either of the preceding parameters is set to a value greater than 1, ultimate RTO is enabled.

Principles

Ultimate RTO makes full use of multi-core CPU resources of servers and uses a multi-level pipeline architecture to implement high availability through replay acceleration. The main principles are as follows: In the upstream of the pipeline, three threads are responsible for reading logs from disks to the cache in the form of byte streams, parsing the byte streams into Xlogs one by one, and parsing and distributing Xlog semantics. In this way, logs are efficiently preprocessed in parallel. The downstream of the pipeline consists of multiple data replay pipelines and one transaction replay pipeline, implementing parallel replay of data page logs and transaction logs. Data page logs are split and distributed to different data replay pipelines by table, and then split and distributed to different replayer threads for replay by page. During distribution, related hash algorithms are used to ensure that logs of the same table are replayed by the same pipeline and logs of the same page on the table are replayed by the same data replayer thread. In this way, the concurrency of standby node log replay is improved to the page level, and the number of pipelines and replayer threads can be increased by setting parameters to improve the replay speed and concurrency. The architecture is shown in **Figure 8-2**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 8-2 Ultimate RTO architecture



Examples

Step 1 Install and deploy the GaussDB cluster.

Step 2 After the database is installed, modify the GUC parameters **recovery_parse_workers** and **recovery_redo_workers** to enable the ultimate RTO replay mode. For details, see "Database GUC Parameters > GUC Parameters > Write Ahead Log > Log Replay" in *Reference*. **recovery_parse_workers** indicates the number of data replay pipelines to be enabled, and **recovery_redo_workers** indicates the number of replayer threads on each data replay pipeline. The setting takes effect only after the database is restarted.

Step 3 Execute the DML service test on the primary node. Then, run the **cm_ctl query -rv** command to query the replay status of the standby node. In the preceding information, the **speed** field indicates the replay speed, and the **delay** field indicates the delay of the standby node.

```
node                      : 3
instance_id               : 6003
node_ip                   : 8.92.5.241
data_path                 : /data/omm/data/slave12
type                      : Datanode
instance_state            : Standby
is_by_query               : 1
primary_flush_location    : 00000023/5D6A4998
standby_received_location : 00000023/5D6A1560
standby_replay_location   : 00000023/5AE12260
static_connections        : 2
HA_state                  : Normal
reason                    : Normal
redo_start_location       : 00000022/5D9EB3D0
min_recovery_location     : 00000023/5569D660
read_location             : 00000023/5D66F6D0
last_replayed_end_location : 00000023/5AE12260
recovery_done_location    : 00000000/00000000
local_max_lsn             : 00000023/5D6743A8
read_xlog_io_counter      : 32253
read_xlog_io_total_dur    : 3840586
read_data_io_counter      : 1649089
read_data_io_total_dur    : 168654452
write_data_io_counter     : 0
write_data_io_total_dur   : 0
process_pending_counter   : 0
process_pending_total_dur : 0
apply_counter             : 5
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
apply_total_dur               : 16095
speed(est.)                   : 78664 KB/s
delay(est.)                   : 0 s
senderPercent                 : 0%
receiverPercent               : 99%
worker_info                   :
id  q_use   q_max_use  rec_cnt
0   42      0          23119337
10  41      0          12575866
```

----End

Precautions

- The ultimate RTO functionality can be enabled by setting the **recovery_parse_workers** and **recovery_redo_workers** parameters. **recovery_parse_workers** indicates the number of enabled replay pipelines, and **recovery_redo_workers** indicates the number of workers in each pipeline. If either of the preceding parameters is set to a value greater than 1, ultimate RTO is enabled.
- After ultimate RTO is enabled, the total number of extra replayer threads started by the standby node = **recovery_parse_workers** × (**recovery_redo_workers** + 2) + 5. More replayer threads occupy more CPU, memory, and I/O resources. Set parameters based on the actual hardware configuration. If the parameter value is too large, the CPU and memory usage may be too high, causing system startup exceptions.

See Also

For details about the parameters related to ultimate RTO, see the descriptions of **recovery_parse_workers** and **recovery_redo_workers** in "Database GUC Parameters > GUC Parameters > Write-Ahead Log > Log Replay" in *Reference*.

8.21 DCF

Description

Concept

Distributed consensus framework (DCF) is a self-developed independent basic library with high performance, high maturity and reliability, easy to expand, and easy to use. Other systems can easily integrate DCF components through APIs. DCF is implemented based on the Paxos protocol to resolve the distributed consistency issue and improve the high reliability and availability of clusters.

Functionalities and benefits

DCF is often used in database HA scenarios to ensure the consistency of multi-copy log storage and provide high availability for external systems. In addition, DCF can automatically elect the leader node, achieving a small RTO when a single point of failure occurs.

Comparison between DCF and Quorum

- Quorum: a mechanism used in distributed systems to ensure data consistency and reliability, mainly used in multi-copy storage systems. It does not support

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

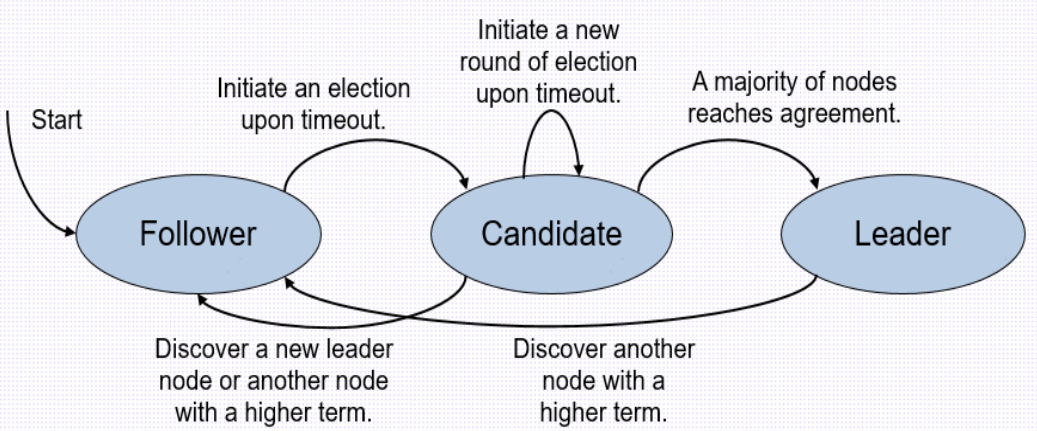
automatic leader election and arbitration. If log forking occurs, build is required for recovery.

- DCF: a distributed consensus framework developed based on the Paxos protocol. It has the logic of automatic leader election and arbitration. When the leader node is faulty, the follower nodes automatically detect the fault and quickly elect the new leader node with a small RTO. Consistency control is performed before operations such as transaction commit and replay. Log forking does not occur when a node is faulty or a switchover is performed, avoiding unnecessary build.

Principles

Basic Principles of DCF Automatic Leader Election

Figure 8-3 Principle of DCF automatic leader election



As shown in **Figure 8-3**, DCF automatic leader election has three basic roles: follower, candidate, and leader.

Timeout-driven: The leader sends a heartbeat message every heartbeat timeout. If followers do not receive heartbeat messages from the leader within the election timeout period, the followers initiate an election and become candidates.

Random start: A random value is added to the election timeout period to reduce the probability of election collision.

Election action: The term increases, the voting message is broadcast, and followers become candidates. The system determines whether to vote based on logs and elects the leader if a candidate obtains more than half of the votes.

Correctness: Only one leader can be formed for the same term. Strict mathematical proof is used to prevent split-brain.

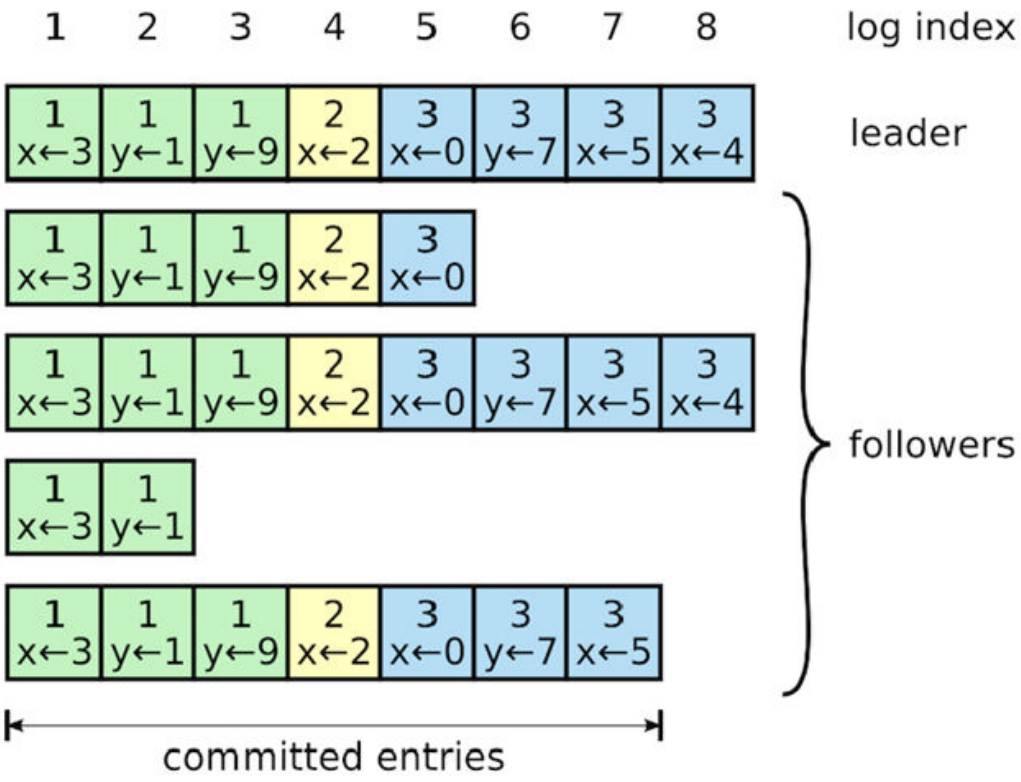
NOTE

A term is equivalent to the logical clock of a distributed system. It is used to handle issues such as leadership contention between nodes, log consistency verification, and expired information identification.

Basic Principles of DCF Log Replication

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

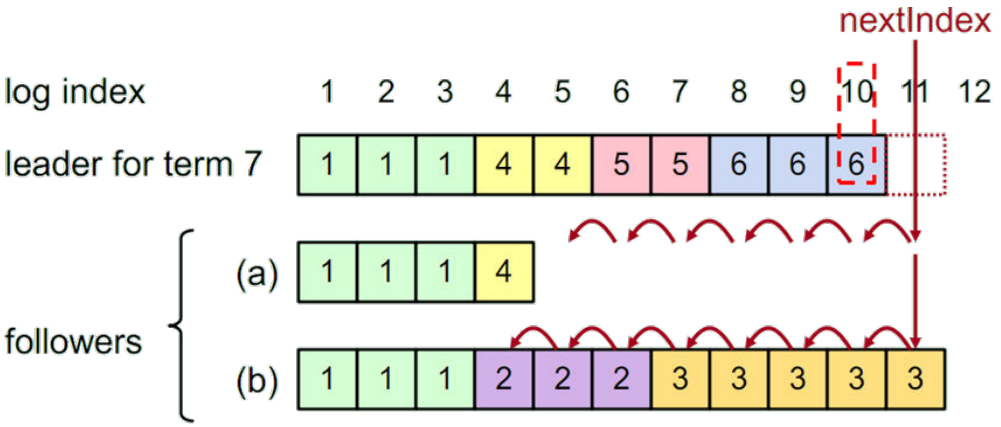
Figure 8-4 DCF log replication principle 1



As shown in **Figure 8-4**, the leader node generates logs and replicates them to the follower nodes. After the majority of logs are consistent, they are committed. Finally, the logs on each node are the same, ensuring strong consistency.

- Each log contains log term and log index.
- If a value is written to the logs on most nodes and in the current term, the logs can be committed. (For the concept of term, see the preceding description.)
- Committed logs cannot be modified and can be applied to upper-layer callers.

Figure 8-5 DCF log replication principle 2



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

When the logs on the leader and follower nodes do not match, DCF automatically rectifies the fault. **Figure 8-5** shows three nodes, a leader node for the term of **7**, and two follower nodes with different log states.

For **follower(a)**, logs are missing after the index of **4** and from indexes 1 to 4, logs are the same as those on the leader node during its term. The leader node synchronizes logs to **follower(a)** from the initial **nextIndex** of **10**. However, **follower(a)** does not have the logs at this location and returns a message indicating the logs do not match. After receiving the response, the leader node decreases **nextIndex** by 1 (to **9**) and continues the synchronization process. When detecting that **nextIndex** is **5**, the leader node synchronizes subsequent logs to **follower(a)**.

The logs on **follower(b)** conflict with those on the leader node. The leader node continuously synchronizes logs to **follower(b)** from **nextIndex** of **10**. Until **nextIndex** is **3**, the logs on this follower node are the same as those on the leader node during its term. At this time, the leader node synchronizes its logs to **follower(b)** from **nextIndex** of **4** and overwrites the conflicting logs after **index** of **4** on **follower(b)**.

The process can be summarized as follows:

- The leader finds the start point of the follower log mismatch.
- The mismatched node logs are rewritten.
- Finally, the logs on all nodes are consistent.

Examples

- The DCF profile log stores the following information about election and replication:

  Figure 8-6 Example of DCF election and replication information

  ```
  [REP] stream_id  node_id    pause     role    leader    term commit_index     match_index              next_index
  [REP]         1        1        0        -         -        - -              (0,0)                             0
  [REP]         1        2        0        -         -        - -              (2,5065)                       5066
  [REP]         1        3        -        1         3        2 (2,5065)       (2,5065)                       5066
  ```

  stream_id: log stream ID. The default value is **1**.

  node_id: node ID

  pause: specifies whether flow control is enabled for a node. The value **0** indicates that flow control is disabled, and a non-zero value indicates that flow control is enabled.

  role: **1**: leader, **2**: follower, **3**: logger, **4**: passive, **5**: pre_candidate, **6**: candidate, **7**: cascade_follower.

  term: term of the current node.

  commit_index: consistency index of the current node.

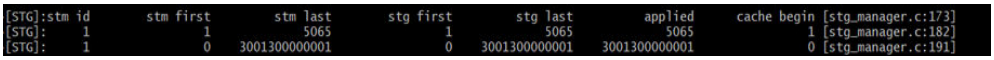
  match_index: index of the DCF log received by the current node.

  next_index: index of the DCF log that the current node expects the leader node to send.

- The DCF profile log stores the following information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 8-7 Example of the information stored by DCF



stm id: log stream ID. The default value is **1**.

stm first: location of the first log in the log stream. The value is updated when dcf_truncate is called.

stm last: location of the last log in the log stream.

stg first: first log stored in the DCF data directory. The value is updated when dcf_truncate is called.

stg last: last log stored in the DCF data directory.

applied: location of the log where the DCF notifies the DN to apply (location where the DN updates the consensus LSN).

cache begin: first log point in the log message cache array of the storage module. After logs are applied or the number of logs reaches the threshold, the logs are recycled to the cache. Generally, the value of **cache begin** is greater than that of **stm first**. **stm last – cache begin** indicates the number of logs stored in the cache array.

See Also

"Database GUC Parameters > GUC Parameters > DCF Parameter Settings" in *Reference*.

8.22 CM

Description

Cluster Manager (CM) is a database resource management and monitoring component. It provides HA and automated instance management, fault recovery capabilities to ensure that clusters can run stably.

GaussDB provides the cm_ctl tool to control cluster services.

Core Components

- **CM Server (cm_server)**
  - It is a component used for managing clusters and arbitrating instances.
  - Description:
    - Receives the status of each instance in the cluster from the CM Agent on each node.
    - Monitors instance status changes and delivers arbitration commands.
    - Provides an API for querying the overall cluster status.
  - HA feature: supports the primary/standby (including one primary and multiple standby nodes) configuration. When a primary CM Server is faulty, a standby CM Server automatically takes over services.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **CM Agent (cm_agent)**
  - It is a cluster management component that is deployed on each host of the cluster to monitor, start, and stop instance processes in each database.
  - Description:
    - Starts, stops, and monitors the database instances on the localhost.
    - Monitors the instance status and reports the status to the CM Server.
    - Executes the arbitration command delivered by the CM Server.
- **OMM (om_monitor)**
  - Monitors the scheduled task component of the CM Agent.
  - Description:
    - Monitors the CM Agent process and periodically checks its running status.
    - Automatically restarts the CM Agent process when it stops to ensure continuous monitoring.
- **Distributed database (DDB)**
  - Management service component used for CM Server automatic arbitration and leader election and persistent storage of arbitration data.
  - CM Server can use ETCD or DCC.
    - ETCD: a highly consistent distributed key-value storage component.
    - DCC: a distributed configuration center component implemented based on the distributed consensus framework (DCF). The DCF uses the Paxos protocol to ensure HA and consistency.

Advantages

- **Unified management**

  Provides unified startup, stopping, query, and switchover functionalities for clusters, nodes, and instances, improving management efficiency.
- **Real-time monitoring**

  Monitors the running status of resources such as nodes, instances, network communication, and file systems from multiple dimensions and detects problems promptly.
- **Flexible expansion**

  Supports customized resource monitoring and fault detection rules to adapt to different service scenarios.
- **Automatic O&M**

  Provides capabilities such as automatic primary/standby failover, reducing the burden of manual O&M.
- **HA assurance**

  The primary/standby CM Server architecture and automatic failover prevent single points of failure and improve system availability.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

- Query the cluster status.

```
cm_ctl query -Cvd

[  CMServer State   ]

node              instance                                    state
------------------------------------------------------------------------------
1  kwephis32851484 1    /var/standard/usr/local/cm/cm_server/cm_server Primary

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL

[ Coordinator State ]

node              instance                    state
------------------------------------------------------------
1  kwephis32851484 5001 /var/standard/usr/local/cn Normal

[ Central Coordinator State ]

node              instance                    state
------------------------------------------------------------
1  kwephis32851484 5001 /var/standard/usr/local/cn Normal

[     GTM State     ]

node              instance                     state
------------------------------------------------------------
1  kwephis32851484 1001 /var/standard/usr/local/gtm P Primary
[  Datanode State   ]

node              instance                                     state
-----------------------------------------------------------------------------------------------
1  kwephis32851484 6001 /var/standard/var/lib/engine/data1/data/datanode1 P Primary Normal
1  kwephis32851484 6002 /var/standard/var/lib/engine/data1/data/datanode2 P Primary Normal
```

- Check the running status of the process.

```
-- Check the running status of the gaussdb process. /var/standard/var/lib/engine/data1/data/datanode1 is the data directory of the specified instance. You can run the cm_ctl query -Cvd command to query the data directory.
cm_ctl check -B gaussdb -T /var/standard/var/lib/engine/data1/data/datanode1

-- Run the following command to check the return value of the previous command. The value 2 indicates that the process exists.
echo $?
2
```

- Set a CM parameter.

```
-- Set the value of the CM Agent parameter log_file_size to 20.
cm_ctl set --param --agent -k "log_file_size=20"
cm_ctl: set cm_agent.conf success.

-- Dynamically load the CM Agent parameter.
cm_ctl reload --param --agent
cm_ctl: reload cm_agent.conf success.
```

- Query CM parameters.

```
cm_ctl list --param --agent

[conf of node(1)]
log_dir = /var/standard/var/log/Mike/Mike/cm/cm_agent
log_file_size = 20
log_min_messages = WARNING
incremental_build = on
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
security_mode = off
upgrade_from = 0
alarm_component = /opt/huawei/snas/bin/snas_cm_cmd
......
```

See Also

"Log Reference > System Logs" in *Reference*

"Database GUC Parameters > CM Parameters" in *Reference*

8.23 Call Stack

Description

A call stack is a core mechanism co-managed by the OS and compiler runtime during program execution. Built upon the stack frame structure and hardware-level stack pointer, it dynamically tracks the function call hierarchy and execution state. In database O&M scenarios, the full-chain visualization capability of the call stack, such as the gs_stack() function in GaussDB, enables real-time capture of thread-level function call chains, accurately pinpointing high-complexity issues such as deadlocks and transaction blocking. Compared with traditional GDB-based analysis, this approach offers millisecond-level response, thread-level fine-grained diagnosis, and uninterrupted service, significantly improving O&M efficiency and reducing fault rectification time.

NOTE

The differences between the data structure stack and the call stack are as follows:

- Data structure stack: A pure logical model based on the Last In, First Out (LIFO) principle, providing only basic operations such as push and pop, with no runtime context management.
- Call stack: A physical-memory implementation that, beyond basic stack operations, relies on compiler-generated stack frame layouts, contiguous memory allocated by the OS, and registers (such as ESP) to dynamically track the call hierarchy. It also integrates context saving and restoration mechanisms to directly support complex program behaviors such as function nesting and recursion.

Principles

The call stack is built upon the stack frame mechanism and hardware-level stack pointer, and is collaboratively managed by the OS and compiler during program execution. Each function call creates an independent stack frame in the runtime stack to store the function's local variables, formal parameters, return address, and call context. When a function completes execution, its corresponding stack frame is automatically popped and control is returned to the caller, enabling seamless state recovery. At the implementation level, the stack typically uses a contiguous memory region (such as an array) or a dynamic linked-list structure, maintaining a pointer to the top of the stack (such as the ESP register in the x86 architecture) to support efficient push and pop operations. In addition, modern processors provide dedicated instructions (such as PUSH and POP) to further enhance the performance and reliability of stack operations. In recursive calls, each level of recursion adds a new stack frame until the termination condition is reached, naturally supporting the nesting and backtracking logic of function calls.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Leveraging the accurate capture capability of the runtime stack, GaussDB introduces the gs_stack() function and gs_ctl stack tool to build a full-chain call stack visualization mechanism spanning from the SQL layer down to the kernel thread level. Built upon the OS's thread scheduling and signal handling framework, this mechanism dynamically captures the function call trace of any thread while the database is running, enabling rapid diagnosis of complex issues such as deadlocks, hangs, and long transaction blocking. The core principle is as follows: given that signal registration (for example, SIGURG) is in place and the target thread is in a responsive state, the mechanism leverages debug symbol tables and stack frame backtracing to reconstruct the underlying assembly call chain into a readable function call path. It supports a maximum stack depth of 128 frames, covering complex nested call scenarios typical in production workloads.

Compared with call-chain analysis methods that rely on GDB (such as the pstack tool), this mechanism offers three key advantages. First, it delivers high real-time performance. The current runtime state can be captured without interrupting services or altering the parent process state. Second, it provides strong environment compatibility, being integrated as a native database tool that requires no additional dependencies. Third, it supports flexible O&M granularity. Call stacks of multiple threads can be retrieved in batches via SQL commands, eliminating the need for manual per-thread operations, while transaction context is automatically correlated to reduce operational complexity.

In addition, GaussDB fully considers production environment constraints in its implementation: it retains original symbol information (non-stripped) to ensure call stack readability; limits concurrent retrieval to prevent additional system load; and provides a unified output format in multi-threaded scenarios to facilitate integration with automated analysis tools. This mechanism has become a critical foundation for database kernel fault diagnosis, enabling O&M personnel to identify root causes from observed symptoms within milliseconds and significantly reducing the mean time to repair (MTTR).

Examples

For details about the gs_stack() function, see "SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.

- Use SELECT * FROM gs_stack(pid) to obtain the call stack of a specified thread.

```
gaussdb=# SELECT * FROM gs_stack(140518384408320); --- Obtain the call stack of the thread indicated by the specified PID.
                                gs_stack
-------------------------------------------------------------------------
 unknown_symbol                                                         +
 poll + 0x2d                                                           +
 WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x307             +
 sys_logger_main(int) + 0xfb4                                          +
 int gauss_db_thread_main<(knl_thread_role)17>(knl_thread_arg*) + 0x5ab+
 internal_thread_func(void*) + 0x2d                                     +
 ThreadStarterFunc(void*) + 0x74                                        +
 unknown_symbol                                                        +
 __clone + 0x6d                                                       +

(1 row)
```

- Use SELECT * FROM gs_stack() to obtain the call stacks of all threads.

```
gaussdb=# SELECT * FROM gs_stack(); --- Obtain the call stacks of all threads.
      tid       | lwtid |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
stack
-----------------+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------
 140519906720128 |  6750 |
unknown_symbol
           +
              |      | poll +
0x2d                                                                                                                                          +
              |      | CommWaitPollParam::caller(int (*)(pollfd*, unsigned long, int), unsigned long) +
0x34                                                  +
              |      | int comm_socket_call<CommWaitPollParam, int (*)(pollfd*, unsigned long,
int)>(CommWaitPollParam*, int (*)(pollfd*, unsigned long, int)) + 0x28          +
              |      | comm_poll(pollfd*, unsigned long, int) +
0xb1                                                                             +
              |      | server_loop() +
0xa33                                                                                                    +
              |      | postmaster_main(int, char**) +
0x4214                                                                                 +
              |      | main +
0x53c                                                                                                         +
              |      | __libc_start_main +
0xf5                                                                                                  +
              |      |
unknown_symbol
           +
              |      |
 140518183073536 |  6817 |
unknown_symbol
           +
              |      | poll +
0x2d                                                                                                          +
              |      | WaitLatchOrSocket(Latch volatile*, int, int, long) +
0x307                                                                +
              |      | WaitLatch(Latch volatile*, int, long) +
0x2e                                                                    +
              |      | AlarmCheckerMain() +
0x2e1                                                                                  +
              |      | int gauss_db_thread_main<(knl_thread_role)20>(knl_thread_arg*) +
0x612                                                              +
              |      | internal_thread_func(void*) +
0x2d                                                                                +
              |      | ThreadStarterFunc(void*) +
0x74                                                                                  +
              |      |
unknown_symbol
           +
              |      | __clone +
0x6d                                                                                                              +
              |      |
 140518384408320 |  6816 |
unknown_symbol
           +
              |      | poll +
0x2d                                                                                                          +
              |      | WaitLatchOrSocket(Latch volatile*, int, int, long) +
0x307                                                                +
              |      | sys_logger_main(int) +
0xfb4                                                                                    +
--More--
```

Precautions

gs_ctl stack -D implements communication based on the SIGUSR1 signal. During the GaussDB process startup, if the SIGUSR1 signal processing function has not been registered, calling gs_ctl stack -D externally will cause the GaussDB process

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

to use the default SIGUSR1 processing function. As a result, the process exits directly.

See Also

"SQL Reference > Functions and Operators > Statistics Information Functions" in *Reference*.

8.24 Wait Events

Description

A wait event refers to the state in which a process or thread in a database cannot continue to execute temporarily because some resources are unavailable or some conditions need to be met.

A transaction or query enters the waiting state when it needs to obtain a lock, read or write data from or to a disk, wait for a response from a remote node, or access a shared memory structure. Although these waits are short, if they occur frequently or take too long each time, the overall throughput will be significantly slowed down. Traditional performance monitoring can only detect that the execution of an SQL statement is slow, but cannot locate the cause. To solve this, the wait event functionality comes into being.

GaussDB introduces a comprehensive wait event system. It not only records whether a wait event occurs, but also accurately collects statistics on the wait durations, number of wait times, and maximum, minimum, and average wait duration. In addition, it supports aggregation analysis by instance, thread, or SQL statement, greatly improving system observability and problem locating efficiency. In general, when the customer's job does not meet the latency requirements or customer expectations, the wait event functionality helps O&M personnel and developers understand the overall system and the detailed execution status of each thread, facilitating exception locating and system optimization.

Principles

The design concept of wait events is to regard wait behavior as traceable and aggregatable time segments. Wait events have nesting relationships. Therefore, the stack structure is used to automatically report the nesting relationships and analyze the cause behind. The recording and statistics of all wait events depend on the local wait event stack of a thread. Each backend thread maintains such a stack structure during running to trace the current wait link. **Figure 8-8** shows the principle of the wait event functionality.

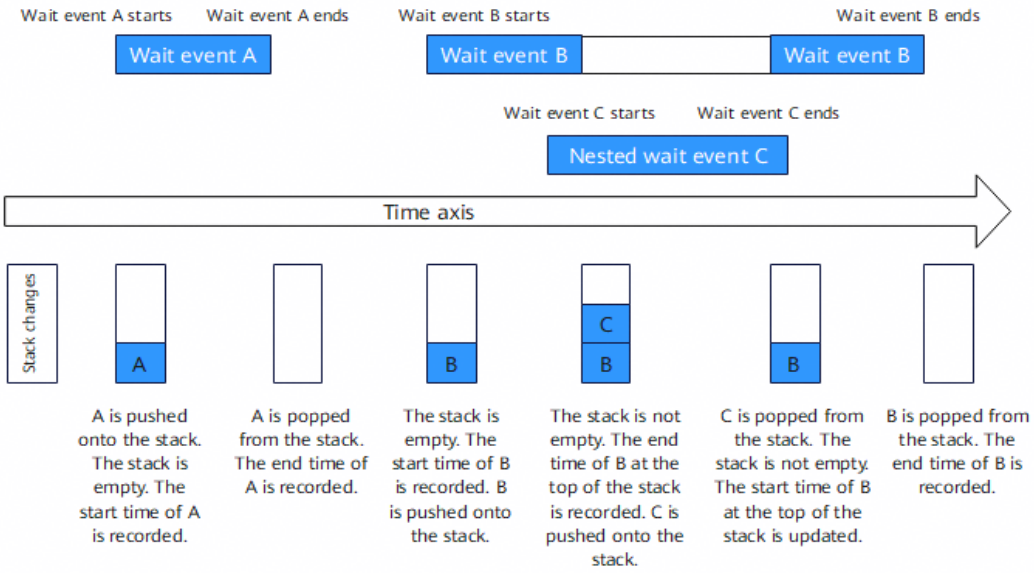
When a thread enters an operation that requires waiting (for example, applying for a lock, reading or writing a file, or waiting for a remote response), the system pushes the wait event to which the current operation belongs to the top of the stack. If there are other wait events in the stack, the system records information such as the start time and wait duration of the wait event at the top of the stack and adds the information to the global statistics view.

When the wait event ends and the thread exits the waiting state, the wait event at the top of the stack is popped out, and the system records the total duration of

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the event. If there are still remaining events in the stack, the system updates the start time of the wait event at the top of the stack to the current time, indicating that the wait event is still ongoing, thus maintaining the continuity of the wait chain.

Figure 8-8 Principle of wait events



NOTE

Wait events are classified into the following types:

- STATUS: Wait for executing a key code segment. This is usually caused by time-consuming synchronization operations or resource coordination, for example, waiting for the application end to send data (STATUS/wait cmd), establishing a connection between the pooler CNs and DNs (STATUS/pooler create conn), obtaining a snapshot from the GTM, and executing SQL operators.
- IO_EVENT: Wait for I/O operations. The values of such wait events are large when a query involves scanning a large amount of data or accessing a large table. Typical scenarios include reading and writing HEAP files and writing WALs.
- LWLOCK: Wait for lock objects that protect the memory structure, which are used to protect globally shared memory structures, such as globally shared hash tables. Such wait events are usually generated in scenarios such as waiting for partition locks.
- LOCK: Wait for internal object locks of the database, such as relation locks, tuple locks, and transaction locks. The values of such wait events are large when a large number of concurrent updates occur.

Examples

- View the statistics about wait events on the current node.

  For details about how to use WAIT_EVENTS, see "Schema > DBE_PERF Schema > Wait Events > DBE_PERF.WAIT_EVENTS" in *Reference*.

```
-- View the statistics about wait events on the current node, including the total wait duration (total_wait_time) and average wait duration (avg_wait_time) of each type of wait event.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT nodename, type, event, wait, total_wait_time, avg_wait_time, last_updated FROM dbe_perf.wait_events ORDER BY total_wait_time DESC;
-[ RECORD 1 ]---+-------------------------------------------
nodename        | coordinator1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
type            | STATUS
event           | wait node
wait            | 393763
total_wait_time | 342584597958
avg_wait_time   | 870027
last_updated    | 2025-09-08 15:49:07.170297+08
-[ RECORD 2 ]---+-----------------------------------------
nodename        | coordinator1
type            | STATUS
event           | wait cmd
wait            | 351923
total_wait_time | 1438714213
avg_wait_time   | 4088
last_updated    | 2025-09-08 15:49:08.257995+08
-[ RECORD 3 ]---+-----------------------------------------
--More--
```

- View the waiting and blocking status of each thread on the current node.

  For details about how to use PG_THREAD_WAIT_STATUS, see "System Catalogs and System Views > System Views > Other System Views > PG_THREAD_WAIT_STATUS" in *Reference*.

```
-- View the waiting and blocking status of each thread on the current node. The wait_status field indicates the type of the event that the current thread is waiting for. For example, if the value of wait_status is acquire lwlock, the current thread is waiting for a lightweight lock or transaction lock.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT node_name, db_name, thread_name, tid, query_id, sessionid, lwtid, wait_status FROM pg_thread_wait_status;
-[ RECORD 1 ]---------------------------
node_name   | coordinator1
db_name     |
thread_name | jit compile worker
tid         | 140601850853120
query_id    | 0
sessionid   | 140601850853120
lwtid       | 56358
wait_status | none
-[ RECORD 2 ]---------------------------
node_name   | coordinator1
db_name     |
thread_name | PageWriter
tid         | 140603134310144
query_id    | 0
sessionid   | 140603134310144
lwtid       | 56261
wait_status | none
-[ RECORD 3 ]---------------------------
--More--
```

- View the wait events of FULL/SLOW SQL statements during execution.

  For details about how to use STATEMENT_HISTORY, see "System Catalogs and System Views > System Catalogs > Other System Catalogs > STATEMENT_HISTORY" in *Reference*.

```
-- View all wait events and their durations during the execution of full/slow SQL statements.
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT query, unique_query_id, pg_catalog.statement_detail_decode(details, 'plaintext', true) FROM pg_catalog.statement_history;
-[ RECORD 1 ]---+-----------------------------------------------------------query          | select gs_perf_do_shaking_collect(?);
unique_query_id | 347666798
detail          |         ----------------Wait Events Area-------------------
                | '1'    STATUS    wait xact start command      139 (us)
                | '2'    STATUS    gtm get snapshot             125 (us)
                | '3'    STATUS    wait xact commit command     121 (us)
                | '4'    IO_EVENT  BufHashTableSearch           26 (us)
                | '5'    STATUS    flush data                   11 (us)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-[ RECORD 2 ]---+-----------------------------------------------------------
--More--
```

Precautions

All wait event statistics are stored in the cumulative view in the memory, which is in the running state. After a node is restarted, the statistics are cleared.

8.25 Backup and Restoration

Description

Backup is a process of copying database files to a specified backup location. It is used to prevent data loss caused by hardware faults, accidental deletion, and ransom attacks, ensuring data security. Restoration is a process of restoring backup data to the database system.

Backup methods:

- Physical backup: copies physical database files to a specified backup location without involving the internal logical structure of the database. It is fast, but the backup files are not readable. This method is suitable for large-scale databases.
- Logical backup: exports logical objects (such as tables, indexes, and triggers) from a database in a logically associated format. The backup files are easy to read, but it is slow.

Restoration methods:

- Physical restoration: copies the physical data files in the original database to a new database.
- Logical restoration: imports the data files (SQL statements or readable text) exported from a database back to the database.

Advantages of backup and restoration:

- Data security

Prevent data loss caused by hardware faults, misoperations, or ransom attacks, ensuring data integrity.

- Business continuity

Reduce downtime through quick restoration, meeting the high availability requirements of industries such as finance and healthcare.

- Compliance

Supports long-term data retention and audit, complying with industry regulations.

Examples

Perform a full physical backup for a database.

```
-- Make preparations.
cd $GPHOME/script # Switch to the Roach tool path.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Query the cluster status.
cm_ctl query -Cv
[  CMServer State   ]

node            instance state
---------------------------------
1  pekphis351063 1        Primary
2  pekphis351064 2        Standby
3  pekphis351065 3        Standby

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : Yes
current_az      : AZ_ALL
...
Backup is meaningful only when the cluster is normal. The preceding command output shows that the cluster status is normal.

-- Log in to the database and write data.
gaussdb=# create table test(id int);
CREATE TABLE
gaussdb=# insert into test values (GENERATE_SERIES(1,100000));
INSERT 0 100000

-- Start the backup.
python3 GaussRoach.py -t backup --master-port 6000 --media-destination /home/userA/media --media-type DISK --metadata-destination /home/userA/metadata
Parsing the configuration file.

Roach master is ready to accept connections. Starting agents now...
Performing presetup activities...

N0 pekphis351063: 100% COMPLETED
N1 pekphis351064: 100% COMPLETED
N2 pekphis351065: 100% COMPLETED

[MASTER]Backup operation SUCCESSFUL. Backup key: 20250819_161344
[MASTER] Time taken : 26s

Successfully send signal to kill all gs_roach.
Cleaning temp files after backup or restore.
Successfully cleaned temp files after backup or restore.
Roach operation backup completed.
```

Users need to specify the required parameters for the backup.

Perform a full physical restoration for a database.

```
-- Restore a database.
python3 GaussRoach.py -t restore --clean --master-port 6000 --media-destination /home/userA/media --media-type DISK --backup-key 20250819_161344 --metadata-destination /home/userA/metadata
Cleaning cluster.
Parsing the configuration file.
Stopping cluster.
Successfully stopped cluster.
Cleaning data before restoration.
Successfully cleaned data before restoration.
Successfully cleaned cluster.

Roach master is ready to accept connections. Starting agents now...
Performing presetup activities...

N0 pekphis351063: 100% COMPLETED
N1 pekphis351064: 100% COMPLETED
N2 pekphis351065: 100% COMPLETED

[MASTER] Restore SUCCESSFUL.
[MASTER] Time taken : 17s
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Cleaning temp files after backup or restore.
Successfully cleaned temp files after backup or restore.
Roach operation restore completed.
```

Users need to specify the required parameters for the restoration.

```
-- Start the database after the restoration is complete.
python3 GaussRoach.py -t start
Stopping the cluster.
Successfully stopped the cluster.
Pre-cleaning ETCD before 'restore-new-cluster' startup...
Successfully pre-cleaned ETCD.
Starting the cluster.
Starting DN master.
Successfully started DN master.
Starting CN.
Successfully started CN.
Starting ETCD.
Successfully started ETCD.
Starting GTM.
Successfully started GTM.
Starting build DN standby.
Successfully start DN standby.
Starting DN dummy standby.
Successfully start DN dummy standby.
Updating catalog phase 1.
Successfully updated catalog phase 1.
Stopping DN instances.
Successfully stop DN instances.
Starting DN master.
Successfully started DN master.
Starting DN standby.
Successfully start DN standby.
Starting CN.
Successfully started CN.
Starting CN.
Successfully started CN.
Updating catalog phase 2.
Successfully updated catalog phase 2.
Starting CN.
Successfully started CN.
Stopping DN instances.
Successfully stop DN instances.
Starting the cluster.
Successfully started primary instance. Please wait for standby instances.
.
Successfully started standby instances.
Waiting for the cluster become normal...
Stopping the cluster.
Successfully stopped the cluster.
Starting the cluster.
Successfully started primary instance. Please wait for standby instances.
.
Successfully started standby instances.
=============================================================================
cluster_state      : Normal
redistributing     : No
node_count         : 3
Coordinator State
    normal         : 3
    abnormal       : 0
GTM State
    primary        : 1
    standby        : 2
    abnormal       : 0
    down           : 0
Datanode State
    primary        : 3
    standby        : 6
    secondary      : 0
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
   building       : 0
   abnormal       : 0
   down           : 0
Successfully started the cluster.
Cleaning backup slot after restore.
Roach operation start completed.
-- Log in to the database again to check whether the data is complete.
gaussdb=# \d
                                List of relations
 Schema | Name | Type  | Owner |                          Storage
--------+------+-------+-------+-------------------------------------------------------------------
 public | test | table | zwb2  | {orientation=row,compression=no,storage_type=USTORE,segment=off}
(1 row)

-- Query data.
gaussdb=# select count(*) from test;
 count
--------
 100000
(1 row)
```

See Also

For more operations related to backup and restoration, see:

"Tool Reference > Data Import and Export Tools" in *Reference*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9 FAQ - Non-Multi-primary

9.1 Database System Overview

9.1.1 Which Hardware Platforms Are Supported by GaussDB?

Answer: GaussDB can run on the following hardware platforms:

- x86_64/Arm-based universal PC server.
- Local storage (SATA, SAS, SSD, and EVS).
- 1-gigabit and 10-gigabit Ethernet. The network must be stable in the database operating environment. It is recommended that the network latency be 50 μs to 70 μs when the cluster load is light (CPU usage is less than 5%) and be less than or equal to 100 μs when the cluster load is heavy (CPU usage is greater than 90%).

9.1.2 How Do I Check the Isolation Level of the Current Database?

Answer: There are the following three transaction isolation levels:

- **READ COMMITTED** (default): At this isolation level, a transaction can access only committed data.
- **REPEATABLE READ**
- **Serializable**: Currently, this isolation level is not supported in functions. It is equivalent to **REPEATABLE READ**.

Run the following statement to check the isolation level of the database:

```
gaussdb=# SHOW transaction_isolation;
 transaction_isolation
-----------------------
 read committed
(1 row)
```

9.1.3 How Easy-to-Use Is GaussDB?

Answer: GaussDB provides various tools and APIs for customers to perform secondary development and integration. Developers can easily use GaussDB to

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

store and process data. In addition, GaussDB provides an easy-to-use operation API and management tool, reducing the usage threshold and learning costs for customers.

9.1.4 How Do I Create and Delete a Savepoint in the Current Transaction of GaussDB?

Answer: A savepoint is a special mark in a transaction. It allows all commands that are executed after the savepoint was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint. SAVEPOINT establishes a new savepoint in the current transaction. RELEASE SAVEPOINT destroys a savepoint previously defined in the current transaction.

For details, see the following example:

```
-- Create a schema and table1.
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO tpcds.table1 VALUES (3);

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
gaussdb=# COMMIT;

-- Drop the schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

9.1.5 What Are the Functions of a Transaction?

Answer: In a database management system, a transaction is used to manage a group of database operations as an inseparable unit of work. These operations are either all successfully executed or all rolled back. A transaction ensures the consistency and integrity of the database and the database can be restored to the previous state, preventing accidents or errors.

The functions of a transaction are as follows:

- Data consistency: All operations in a group in a transaction are successfully executed or rolled back, ensuring data consistency in the database.
- Data integrity: Transactions use the atomicity, consistency, isolation, and durability (ACID) attributes to ensure data integrity, preventing data loss or damage.
- Isolation: Transaction isolation ensures that multiple concurrent transactions are isolated from each other, preventing cross data access and interference and ensuring the independence of each transaction.
- Atomicity: All transactions are successfully executed or rolled back. This atomicity ensures transaction integrity.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Durability: Once a transaction is committed, its modifications to the database are permanently saved.

9.1.6 How Do I Start a Transaction?

Answer: You can start a transaction using either BEGIN or START TRANSACTION.

- Usage of BEGIN.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test1(col1 int, col2 int);
gaussdb=# INSERT INTO tbl_test1 VALUES (1,1), (2,2), (3,3);

-- Start a transaction in default mode.
gaussdb=# BEGIN;
INSERT INTO tbl_test1 VALUES (4,4);
END;

-- Drop the table.
gaussdb=# DROP TABLE tbl_test1;
```

- Usage of START TRANSACTION.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason (c1 int, c2 int);

-- Start a transaction in default mode.
gaussdb=# START TRANSACTION;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

-- Drop the tpcds.reason table.
gaussdb=# DROP TABLE tpcds.reason;

-- Drop the schema.
gaussdb=# DROP SCHEMA tpcds;
```

9.1.7 What Are the Four Features of Correctly Executing Database Transactions?

Answer: During data write or update, the database management system (DBMS) must meet the following requirements: Atomicity, Consistency, Isolation, and Durability (ACID) to ensure that transactions are correctly executed.

- Atomicity: also called indivisibility. All operations in a transaction are either completed or not completed, and do not end in an intermediate phase. If an error occurs during transaction execution, the transaction is rolled back to the state before the transaction starts, just as if the transaction has never been executed.
- Consistency: The database integrity is not damaged before and after a transaction starts. This means that the written data must fully comply with all preset rules, including the accuracy of the data, the serialization of the data, and the spontaneous completion of the scheduled work by the subsequent database.
- Isolation: also called independence. Transactions are often executed concurrently. Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Transaction isolation is divided into different levels, including read uncommitted, read committed, repeatable read, and serializable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Durability: Once a transaction has been committed, it will remain committed even in the case of a system failure.

9.1.8 What is Dirty Read, Phantom Read, and Non-repeatable Read?

Answer: Dirty read, phantom read, and non-repeatable read are problems of concurrent transactions.

- Dirty read: Transaction A reads data that has been modified by transaction B but has not been submitted, and transaction B is rolled back.
- Phantom read: If the same data is read in two consecutive periods, inconsistent results may occur.
- Non-repeatable read: The data queried twice in the same transaction is different.

9.1.9 What Is MVCC?

Answer: MVCC refers to Multi-Version Concurrency Control. MVCC is a database concurrency control protocol. A write transaction does not modify the tuple content immediately. Each operation creates a target version based on the source version and retains the source version. When a transaction needs to read data, the database system selects a version that meets the transaction isolation level requirement from all versions.

A main advantage of the MVCC is that MVCC does not cause conflicts between lock requests for data reading and writing and keep them unaffected by each other.

9.2 Database Connection Management

9.2.1 How Do I Query the Number of Active Database Connections?

Answer: Set **track_activities** to **on** and query the pgxc_stat_activity view.

```
-- Set the parameter track_activities to on.
gaussdb=# SET track_activities = on;

-- Query the number of connections.
gaussdb=# SELECT COUNT(*) FROM pgxc_stat_activity WHERE state='active';
 count
-------
    19
(1 row)
```

9.2.2 What Are the Causes of the Failure to Connect to the Database Through the Client?

Answer: When using the database, you need to connect to the database through the client. You can perform operations in the database only when you correctly connect to the database. If you cannot connect to the database, the common causes are as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The number of database connections has reached the maximum allowed limit.
- Users do not have the access permission to the database.
- The client IP address is not configured in the **pg_hba.conf** file of the database server.
- The network connection is abnormal.

9.2.3 What Client Access Authentication Modes Does GaussDB Support?

Answer: GaussDB supports the following three authentication modes:

- Host-based authentication: A server checks the configuration file based on the IP address, username, and target database of the client to determine whether the user can be authenticated.
- Password authentication: A password can be an encrypted password for remote connection or a non-encrypted password for local connection.
- SSL encryption: OpenSSL provides a secure connection between the server and the client.

The **gs_hba.conf** file must be configured for all the preceding authentication modes. In the **gs_hba.conf** file, each record occupies one row and specifies an authentication rule. An empty row or a row started with a number sign (#) is neglected. Each authentication rule consists of multiple columns separated by spaces and forward slashes (/), or spaces and tab characters. If a column is enclosed with quotation marks ("), it can contain spaces. A record cannot span different rows.

9.2.4 How Do I Check the Number of Database Application Connections and the Maximum Number of Connections?

Answer: If the number of connections between applications and the database exceeds the maximum value, new connections cannot be established. Therefore, you need to release idle connections or increase the maximum number of connections in a timely manner to ensure normal connections to database applications.

```
-- Check the number of database application connections.
gaussdb=# SELECT count(*) FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
 count
-------
    65
(1 row)

-- Check the maximum number of database connections.
gaussdb=# SHOW max_connections;
 max_connections
-----------------
 1024
(1 row)
```

9.2.5 How do I obtain the username for connecting to a database?

Answer: Run the following command to query the current username:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# SELECT current_user;
```

9.2.6 How Do I Check the Maximum Number of Session Connections for Specific Users?

Answer: To check the maximum number of session connections allowed for specific users, do as follows:

```
-- Check the connection limit for user omm.
gaussdb=# SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='omm';
 rolname | rolconnlimit
---------+--------------
 omm     |           -1
(1 row)
Note: A value of -1 indicates that there is no connection limit for user omm.

-- Check the connection limit for user omm1.
gaussdb=# SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='omm1';
 rolname | rolconnlimit
---------+--------------
 omm1    |           10
(1 row)
Note: The result shows that the connection limit for user omm1 is 10.
```

9.3 Database Security

9.3.1 How Do I Manually Lock and Unlock a User?

Answer: During the use of the database, if an administrator finds that an account has unauthorized access, the administrator can manually lock the account. After determining that the account is restored, the administrator can manually unlock the account.

The following uses manually locking and unlocking user **jack** as an example:

- Manually lock a user.
  ```
  gaussdb=# ALTER USER jack ACCOUNT LOCK;
  ALTER ROLE
  ```
- Manually unlock a user.
  ```
  gaussdb=# ALTER USER jack ACCOUNT UNLOCK;
  ALTER ROLE
  ```

9.3.2 What Security Management Functions Are Supported in GaussDB?

Answer: SSL network connections, user permission management, password management, security auditing, and other functions are supported to ensure data security at the management, application, system, and network layers. In GaussDB Agent mode, HTTPS is used by default for transmission. During transmission in HTTP, data is not encrypted and transmitted in plaintext, posing security risks.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.3.3 How Do I Generate an OBS Transfer Key Before Creating a Model Service?

Answer: When a service is created, the API key is encrypted. The principle is to reuse the key file used for OBS transmission. If the file does not exist, run the following command to generate it:

```
gs_guc generate -S username -D $GAUSSHOME/bin -o obsserver
```

9.3.4 How Does a Dynamic Data Masking Policy Take Effect?

Answer: By default, the security policy of GaussDB is disabled (**off**). Generally, you need to set it to **on** to make all dynamic data masking policies take effect. In addition to the main security switch, each masking policy has a corresponding switch. The corresponding data masking policy takes effect only after its own switch is turned on.

- Turn on the main security switch.

```
-- Set the GUC parameter enable_security_policy to on.
gs_guc reload -Z coordinator -Z datanode -N all -I all -c "enable_security_policy=on"
```

- Set only one data masking policy.

```
-- Enable a data masking policy.
gaussdb=# ALTER MASKING POLICY policy_name enable;
```

9.3.5 How Do I Set the Validity Period of an Account?

Answer: During the use of the database, some users are created to perform related operations on the database. When creating a user, you need to restrict the operation period (that is, the start time and end time of the validity period) of the user. The setting method is as follows:

Step 1 Connect to the database using gsql.

Step 2 Create a user and specify the start time and end time of its validity period.

```
gaussdb=# CREATE USER jack WITH PASSWORD '********' VALID BEGIN '2015-10-10 08:00:00' VALID UNTIL
'2016-10-10 08:00:00';
CREATE ROLE
```

The user is created if the CREATE ROLE information is displayed:

Step 3 If the user is not within the specified validity period, run the following command to set a new validity period, including the start time and end time.

```
gaussdb=# ALTER USER jack WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10 08:00:00';
ALTER ROLE
```

The start time and end time of the new validity period is set successfully if the ALTER ROLE information is displayed:

----End

9.3.6 How Do I Enable Audit Policies on GaussDB and What Contents Can Be Audited?

Answer: The global audit switch, **audit_enabled**, can be loaded dynamically. If you change the parameter value when the database is running, the change takes effect immediately and you do not need to restart the database. The default value is **on**, indicating that the audit function is enabled. In addition to the overall audit

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

configuration, each audit item can be independently configured. The function of each audit item takes effect only after the configuration is enabled. The configuration of each audit item supports dynamic loading. If you change the audit configuration when the database is running, the change takes effect immediately and you do not need to restart the database.

- Enable the audit function.
```
-- Set the GUC parameter audit_enabled to on.
gs_guc reload -Z coordinator -Z datanode -N all -I all -c "audit_enabled = on"
```
- Set only a single audit item.
```
-- Enable the audit of DDL operations on database objects (DATABASE, SCHEMA, USER, NODE GROUP, and SQL Patch).
gs_guc reload -Z coordinator -D /gaussdb/data/coordinator -c "audit_system_object=67121159"
```

NOTE

The default parameter value of each audit item meets security standards. You can enable other audit functions, but doing so may affect the system performance. For details about how to configure audit items, see "Database Security > Database Audit" in *Developer Guide*.

9.3.7 How Do I View the Table Permissions of a User?

Answer: You can use the information_schema.table_privileges system view to view the information.

```
-- Query the tables that user jim has permission on.
gaussdb=# SELECT * FROM information_schema.table_privileges WHERE grantee = 'jim';
 grantor | grantee | table_catalog | table_schema | table_name | privilege_type | is_grantable | with_hierarchy
---------+---------+---------------+--------------+------------+----------------+--------------+----------------
 jim     | jim     | postgres      | jim          | reason     | INSERT         | YES          | NO
 jim     | jim     | postgres      | jim          | reason     | SELECT         | YES          | YES
 jim     | jim     | postgres      | jim          | reason     | UPDATE         | YES          | NO
 jim     | jim     | postgres      | jim          | reason     | DELETE         | YES          | NO
 jim     | jim     | postgres      | jim          | reason     | TRUNCATE       | YES          | NO
 jim     | jim     | postgres      | jim          | reason     | REFERENCES     | YES          | NO
 jim     | jim     | postgres      | jim          | reason     | TRIGGER        | YES          | NO
(7 rows)
```

Table 9-1 describes the columns in the system view information_schema.table_privileges.

Table 9-1 table_privileges columns

Column	Data Type	Description
grantor	information_schema.sql_identifier	Permission grantor.
grantee	information_schema.sql_identifier	Permission grantee.
table_catalog	information_schema.sql_identifier	Database that contains the table.
table_schema	information_schema.sql_identifier	Schema that contains the table.
table_name	information_schema.sql_identifier	Table name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Column	Data Type	Description
privilege_type	information_schema.character_data	Type of the granted permission. The value can be **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCES**, **ANALYZE**, **VACUUM**, **ALTER**, **DROP**, or **TRIGGER**.
is_grantable	information_schema.yes_or_no	Specifies whether the permission can be granted to other users. **YES** indicates that the permission can be granted to other users, and **NO** indicates that the permission cannot be granted to other users.
with_hierarchy	information_schema.yes_or_no	Specifies whether specific operations are allowed to be inherited at the table level. If the specific operation is **SELECT**, **YES** is displayed. Otherwise, **NO** is displayed.

9.3.8 How Do I Enable Separation of Duties for GaussDB?

Answer: To avoid risks caused by centralized permissions, you can enable separation of duties to assign the system administrator's user management permission to security administrators and audit management permission to audit administrators. If the parameter **enableSeparationOfDuty** is set to **on**, the separation of duties is enabled to control system administrator permissions. System administrators are not allowed to create or modify user configurations. In addition, you are advised to disable the GUC parameter **enable_copy_server_files** to control the COPY permission of a system administrator, preventing the system administrator from reading or modifying the user configuration file using the copy command. The configuration method is as follows:

Step 1 Switch to the instance user **omm**.

```
su - omm
```

Step 2 Run the following statement to set **enableSeparationOfDuty** to **on**.

```
gs_guc set -Z coordinator -Z datanode -N all -I all -c "enableSeparationOfDuty = on"
```

Step 3 Restart the database.

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

----End

9.3.9 How Do I Create a Read-only User on GaussDB?

Answer: In service scenarios, database administrators can use schemas to divide services. For example, if you want to create a read-only user **user1**, you can allow this user to access all tables (including new tables created later) in the **sc_test** schema, but cannot insert, modify, or delete data. The detailed process is as follows:

```
-- Preparations: Create a user and the sc_test schema.
gaussdb=# CREATE ROLE u1 WITH SYSADMIN PASSWORD '********';
gaussdb=# CREATE SCHEMA sc_test;
gaussdb=# ALTER SCHEMA sc_test OWNER TO u1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create user user1.
gaussdb=# CREATE ROLE user1 WITH LOGIN PASSWORD '********';

-- Grant user1 the permission to access the sc_test schema.
gaussdb=# GRANT USAGE ON SCHEMA sc_test TO user1;

-- Grant user1 the SELECT permission on all tables in the sc_test schema.
gaussdb=# GRANT SELECT ON ALL TABLES IN SCHEMA sc_test TO user1;

-- Grant user1 the SELECT permission for tables to be created in the sc_test schema.
gaussdb=# ALTER DEFAULT PRIVILEGES FOR USER u1 GRANT SELECT ON TABLES TO user1;

-- Drop the user.
gaussdb=# ALTER DEFAULT PRIVILEGES FOR USER u1 REVOKE SELECT ON TABLES FROM user1;
gaussdb=# DROP SCHEMA sc_test;
gaussdb=# DROP ROLE user1;
gaussdb=# DROP ROLE u1;
```

9.3.10 What Are the Session States of GaussDB? How Do I Query the Session Status of a User?

Answer: There are six states. The details are as follows:

- **active**: The background is executing a query.
- **idle**: The backend is waiting for a new client command.
- **idle in transaction**: The background is in a transaction, but there is no statement being executed in the transaction.
- **idle in transaction (aborted)**: The backend is in a transaction, but a statement in the transaction failed.
- **fastpath function call**: The background is executing a fast-path function.
- **disabled**: This state is reported if **track_activities** is disabled in this backend.

The following is an example of querying the session status of a user:

```
gaussdb=#  SELECT datname, usename, usesysid, state, pid FROM pg_stat_activity;
 datname  | usename | usesysid | state  |       pid
----------+---------+----------+--------+-----------------
 postgres | omm1    |       10 | idle   | 139669908420352
 postgres | omm1    |       10 | active | 139669573990144
```

9.3.11 How Do I Query the Expiration Date of an SSL Certificate?

Answer: The SSL certificate information of the GaussDB instance is saved in *$GAUSSHOME*/**share/sslcert/**. The operations are as follows:

Step 1 Go to the directory where the SSL certificate information is stored.

```
[omm1@node41 ~]$ cd $GAUSSHOME/share/sslcert/
```

Step 2 Go to the directory where the **client.crt** certificate is located.

```
[omm1@node41 ~]$ cd ./gsql && ll
total 56
-rw------- 1 omm1 dbgrp 5690 Apr  7 11:14 cacert.pem
-rw------- 1 omm1 dbgrp 5713 Apr  7 11:14 client.crt
-rw------- 1 omm1 dbgrp 2654 Apr  7 11:14 client.key
-rw------- 1 omm1 dbgrp   56 Apr  7 11:14 client.key.cipher
-rw------- 1 omm1 dbgrp   24 Apr  7 11:14 client.key.rand
-rw------- 1 omm1 dbgrp 4542 Apr  7 11:14 openssl.cnf
-rw------- 1 omm1 dbgrp 5713 Apr  7 11:14 server.crt
-rw------- 1 omm1 dbgrp 2654 Apr  7 11:14 server.key
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-rw------- 1 omm1 dbgrp   56 Apr  7 11:14 server.key.cipher
-rw------- 1 omm1 dbgrp   24 Apr  7 11:14 server.key.rand
```

Step 3 Use the OpenSSL command tool to output the date of the certificate.

```
-- notBefore indicates the start date of the certificate validity period, and notAfter indicates the expiration date of the certificate.
[omm1@node41 gsql]$ openssl x509 -in client.crt -noout -dates
notBefore=Apr  7 03:14:21 2024 GMT
notAfter=Apr  2 03:14:21 2044 GMT
```

----End

9.3.12 How Do I Enable Password Complexity Verification?

Answer: To ensure database security, you are advised to enable password complexity verification so that password complexity is verified when a user is created or a password is changed.

The configuration and verification methods are as follows:

```
-- Configuration method.
gs_guc reload -Z coordinator -N all -I all -c "password_policy=1"

-- Verification method.
gaussdb=# SELECT name,setting FROM pg_settings WHERE name='password_policy';
      name       | setting
-----------------+---------
 password_policy | 1
(1 row)
```

9.3.13 How Do I Check the Initialized User Role of a Database?

Answer: The initialized user role has the highest database permissions, that is, has all system and object permissions. You are advised to use this user only for DBA management.

```
-- Check the initialized user role.
gaussdb=# SELECT rolname FROM pg_roles WHERE rolsuper=true;
 rolname
---------
 omm
(1 row)
```

9.3.14 Which System Views Are Related to Audit Information?

Answer: The following three views are related to audit information and can be accessed only by users with the SYSADMIN or POLADMIN permission:

- GS_AUDITING: displays all audit information about database-related operations.
- GS_AUDITING_ACCESS: displays all audit information about database DML-related operations.
- GS_AUDITING_PRIVILEGE: displays all audit information about database DDL-related operations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.3.15 How Do I Configure Log File Permissions?

Answer: After **logging_collector** is enabled, you can set **log_file_mode** to specify server log file permissions. Access to log files must be restricted because the log files may contain sensitive data.

```
The configuration method is as follows:
-- Run the following statement to set log_file_mode to 0600.
gs_guc reload -Z coordinator -Z datanode -N all -I all -c "log_file_mode=0600"

The check method is as follows:
gaussdb=# SELECT name,setting FROM pg_settings WHERE name='log_file_mode' AND setting != '0600';
```

9.3.16 How Do I Set the Maximum Number of Failed Login Attempts for a Database Account?

Answer: You are advised to set the maximum number of failed login attempts to protect passwords from brute-force attacks. After the parameter **failed_login_attempts** is set, an account will be automatically locked if the number of its failed login attempts exceeds the parameter value.

```
-- Set the GUC parameter failed_login_attempts to 10.
gs_guc reload -Z coordinator -Z datanode -N all -I all -c "failed_login_attempts=10"
```

9.3.17 What Are the Differences Between TDE and Full Encryption?

Answer: Both TDE and full encryption can encrypt data, but they represent two distinct encryption modes. Their key differences are outlined below:

- Different encryption locations: TDE occurs after the database storage module, whereas full encryption is applied within database drivers.
- Different encryption objects: TDE encrypts all data on data pages, whereas full encryption encrypts specific columns in SQL statements.
- Different encryption performance: TDE can process 8 KB data pages at a time, offering higher performance compared to full encryption, which processes only one data record in an SQL statement at a time.
- Different security: Full encryption is more secure than TDE.

9.3.18 How Do We Ensure Data Security?

Answer: GaussDB employs multiple security measures to ensure data security, including:

- Encryption technology: Sensitive data is encrypted during storage and transmission to prevent leakage.
- Access control: User authentication and permission management restrict data access based on users.
- Backup and restoration: Regular data backups enable swift restoration when needed to prevent data loss.
- Audit and monitoring: The system records user operations and monitors and generates alarms in real time to promptly detect and address security events.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.3.19 How Do I Create a Weak Password in GaussDB?

Answer: In the GaussDB database, the weak password dictionary is empty by default. You can manually add one or more weak passwords to the gs_global_config system catalog.

Example:

```
-- Insert a single weak password into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

-- Insert multiple weak passwords into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');

-- Check weak passwords in the gs_global_config system catalog.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
      name       |     value
-----------------+----------------
 weak_password   | password1
 weak_password   | password2
 weak_password   | password3
(3 rows)
```

9.3.20 What Are the Schemas, Users, and Roles in GaussDB?

Answer: A schema is a logical division of a database. All database objects are created under certain schemas. Schemas and users are loosely bound. When you create a user, a schema with the same name as the user will be created automatically. You can also create a schema or specify another schema.

Users and roles are used to control the access to databases. A database may be used by many users, and users are grouped into a database role for easy management. A role can be a database user or a group of database users, depending on role settings. The difference between roles and users is that a role does not have the LOGIN permission by default. In service design, you are advised to use a role to configure permissions rather than accessing databases.

9.3.21 Why Can a User Log In to the Database After the CONNECT ON DATABASE Permission Is Granted to the REVOKE User?

Answer: After you run the **REVOKE CONNECT ON DATABASE dbname FROM u1** command to revoke the u1 permission, you need to specify **public** because the CONNECT permission of the database is granted to **public**.

- GaussDB provides an implicitly defined group **public** that contains all roles. By default, all new users and roles have the permissions of **public**. 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.
- GaussDB grants the permissions on certain types of objects to **public**. By default, permissions on the following objects are granted to **public**:
  - CONNECT permission for databases
  - CREATE TEMP TABLE permission
  - EXECUTE permission for functions
  - USAGE permission for languages and data types (including domains)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- An object owner can revoke the default permissions granted to **public** users and grant permissions to other users as needed.

Example:

1. Create the **u1** user and the **db_test** database.
```
gaussdb=# CREATE USER u1 PASSWORD '********';
gaussdb=# CREATE DATABASE db_test;
```
2. The **u1** user can connect to the **test** database. Log out of the database.
```
gsql -d db_test -h xxx.xxx.xxx.xxx -p xxxx -U u1 -W '********'
```
3. Run the **REVOKE** command to revoke the **PUBLIC** user's permission to connect to **db_test**.
```
gaussdb=# REVOKE CONNECT ON DATABASE db_test FROM public;
```
4. Verify the result and connect to the database as the **u1** user. The result shows that the **u1** user cannot connect to the database.
```
gsql -d db_test -h xxx.xxx.xxx.xxx -p xxxx -U u1 -W '********'
gsql: FATAL:  permission denied for database "db_test"
DETAIL:  User does not have CONNECT privilege.

-- Drop the user and database.
gaussdb=# DROP USER u1;
gaussdb=# DROP DATABASE db_test;
```

9.3.22 How Do I Check the Creation Time of a Database User?

Answer: View the **passwordtime** column in the PG_AUTH_HISTORY system catalog to check the initial setting time of the user password, that is, the creation time of the user.

```
gaussdb=# SELECT t1.usename, t1.usesysid, min(t2.passwordtime) as create_time FROM pg_user t1,
pg_auth_history t2 WHERE t1.usename='jerry' AND t1.usesysid = t2.roloid GROUP BY t1.usename,
t1.usesysid;
 usename | usesysid |          create_time
---------+----------+-------------------------------
 jerry   |    27311 | 2024-07-23 14:46:08.284336+08
(1 row)
```

9.4 Database Quick Start

9.4.1 What are the differences between the DROP, TRUNCATE, and DELETE methods in GaussDB?

Answer: The differences between the DROP, TRUNCATE, and DELETE methods lie in the deletion speed and scope. The details are as follows:

- The DROP statement is the fastest. It can remove an entire table, including the table definition, data, indexes, and permissions.
- The speed of the TRUNCATE statement is medium. It can remove all data in a table but does not delete the table definition.
- The DELETE statement is the slowest. It can remove data from a table based on conditions, excluding the table definition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.4.2 How many bytes does a Chinese character occupy in GaussDB?

Answer: It depends on the character set encoding of the database. For the GBK character set, a Chinese character occupies two bytes. For the UTF-8 character set, a Chinese character occupies three bytes.

9.4.3 How Do I Create Indexes in Parallel?

Answer: Refer to the following method:

```
-- Set the maintenance_work_mem parameter as required.
gaussdb=# SET maintenance_work_mem = '8GB';

-- Create a table.
gaussdb=# CREATE TABLE table_name (col1 int, col2 int);

-- Change the number of threads for creating indexes for a table based on the actual situation.
gaussdb=# ALTER TABLE table_name SET (parallel_workers=4);

-- Create an index.
gaussdb=# CREATE INDEX index_name ON table_name(col1);

-- Reset the parallel_workers parameter for the table.
gaussdb=# ALTER TABLE table_name RESET (parallel_workers);

-- Drop the index.
gaussdb=# DROP INDEX index_name;

-- Drop the table.
gaussdb=# DROP TABLE table_name;
```

9.4.4 How Do I Check the Character Encoding Types of the GaussDB Client and Server?

Answer: Run the following statement to check the character encoding type of the client:

```
gaussdb=# SHOW client_encoding;
 client_encoding
-----------------
 UTF8
(1 row)
```

Run the following statement to check the character encoding type of the server:

```
gaussdb=# SHOW server_encoding;
 server_encoding
-----------------
 SQL_ASCII
(1 row)
```

9.4.5 How Do I View the Current Session ID and Login Time?

Answer: Run the following SQL statements to view the current session ID and login time:

```
-- View the current session ID.
gaussdb=# SELECT pg_backend_pid() AS session_id;
session_id
-----------------
 139883131631360
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- View the login time of the current session.
gaussdb=# SELECT pg_stat_activity.backend_start AS login_time FROM pg_stat_activity WHERE
pg_stat_activity.pid = pg_backend_pid();
 login_time
-------------------------------
 2024-07-19 16:39:45.429437+08
(1 row)
```

9.4.6 How Do I View the Creation Time of a Table?

Answer: See the value of the **created** column in the PG_TABLES system view.

```
-- Create a table.
gaussdb=# CREATE TABLE test(id int, name varchar(10));

-- Query the time when the test table is created.
gaussdb=# SELECT tablename,created FROM pg_tables WHERE tablename = 'test';
 tablename |            created
-----------+-------------------------------
 test      | 2024-01-12 14:50:59.611988+08
(1 row)
```

9.4.7 How Do I View Database Parameter Values?

Answer: To view database parameter, you can run the **SHOW** command or check them in the pg_settings view. For example:

```
-- Method 1: Run the SHOW command to view the value of shared_buffers.
gaussdb=# SHOW shared_buffers;
 shared_buffers
----------------
 1GB
(1 row)


-- Method 2: Check the value of shared_buffers in the pg_settings view.
gaussdb=# SELECT name, setting FROM  pg_settings WHERE name='shared_buffers';
      name      | setting
----------------+---------
 shared_buffers | 131072
(1 row)
```

9.4.8 How Do I View Foreign Table Information?

Answer: You can use the pg_get_tabledef() function to view foreign table information. The following is an example:

```
-- Create a server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;
-- Create a foreign table.
gaussdb=# CREATE FOREIGN TABLE foreign_tbl (col1 text) SERVER my_server OPTIONS (logtype 'gs_log');
-- View information about the foreign table foreign_tbl.
gaussdb=# SELECT * FROM pg_get_tabledef('foreign_tbl');
         pg_get_tabledef
------------------------------------
 SET search_path = public;         +
 CREATE FOREIGN TABLE foreign_tbl (+
    col1 text                     +
 )                                +
 SERVER my_server                 +
 OPTIONS (                        +
    logtype 'gs_log'              +
 )                                +
 DISTRIBUTE BY ROUNDROBIN;
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the foreign table.
gaussdb=# DROP FOREIGN TABLE foreign_tbl;
-- Drop the server.
gaussdb=# DROP SERVER my_server;
```

9.4.9 How Do I View the SELECT Statements That Are Running in the System?

Answer: Set the parameter **track_activities** to **on**. The database system collects the execution information about active queries only if this parameter is set to **on**.

```
gaussdb=# SET track_activities = on;
```

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

```
gaussdb=# SELECT datname, usename, state,pid,query  FROM pgxc_stat_activity WHERE state = 'active';
 datname  | usename | state  |       pid       |                                          query
----------+---------+--------+-----------------
+-------------------------------------------------------------------------------------------
 postgres | ommttjj | active | 281431140520064 |
 postgres | ommttjj | active | 281431621030016 |
 postgres | ommttjj | active | 281430469562496 |
 postgres | ommttjj | active | 281430402191488 | WLM fetch collect info from data nodes
 postgres | ommttjj | active | 281429778288768 | SELECT datname, usename, state,pid,query  FROM
pg_stat_activity where state = 'active';
(5 rows)
```

9.4.10 How Do I Query Indexes in a Table?

Answer: You can query the indexes of a table in the pg_indexes view. Example:

```
-- Query the existing indexes in the tbl_test1 table.
gaussdb=#  SELECT * FROM pg_indexes WHERE tablename='tbl_test1';
 schemaname | tablename | indexname | tablespace |
indexdef

------------+-----------+-----------+------------
+-------------------------------------------------------------------------------------------------------------
 public     | tbl_test1 | idx_test1 |            | CREATE INDEX idx_test1 ON tb1_test1 USING ubtree (a) WITH
(storage_type=USTORE) TABLESPACE pg_default
(1 row)
```

9.4.11 How Do I Query the Storage Mode of a Table?

Answer: You can use the table definition function pg_get_tabledef() to query the table. **row** indicates row store. The query method is as follows:

```
-- Query the storage mode of the test_1 table.
gaussdb=# SELECT * FROM pg_get_tabledef('test_1');
                        pg_get_tabledef
--------------------------------------------------------------------------------
 SET search_path = public;                                                     +
 CREATE TABLE test_1 (                                                         +
     a integer                                                                 +
 )                                                                             +
 WITH (orientation=row, compression=no, storage_type=USTORE, segment=off)+
 DISTRIBUTE BY HASH(a)                                                         +
 TO GROUP group_version1;
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.4.12 How Do I Query the Table and Database Capacity?

Answer: Use the query functions pg_table_size() and pg_database_size().

```
-- Create tables and insert data into the tables.
gaussdb=# CREATE TABLE test(id int, name varchar(10));
gaussdb=# INSERT INTO test VALUES(1,'Tony'),(2,'Thor'),(3,'Hunk');
-- Query the capacity of the test table.
gaussdb=# SELECT pg_table_size('test');
 pg_table_size
---------------
          8192
(1 row)

-- Query the postgres database capacity.
gaussdb=# SELECT pg_database_size('postgres');
 pg_database_size
------------------
        235134980
(1 row)
```

9.4.13 How Do I Query the Number of Sessions?

Answer: You can run the following SQL statement to query the number of sessions:

```
gaussdb=# SELECT state,count(*) FROM pgxc_stat_activity group by 1 order by 2 desc;
 state  | count
--------+-------
 idle   |    30
 active |    22
        |    12
(3 rows)
```

Note: If the value of **state** is null, it indicates that the session belongs to another user.

9.4.14 How Do I Query the Sizes of Databases, Tablespaces, and Tables?

- Query the database size.

  ```
  gaussdb=#  SELECT datname, pg_database_size(datname), pg_database_size(datname)/
  1024/1024/1024 as "dataSize_GB" FROM pg_database;
        datname        | pg_database_size |     dataSize_GB
  ---------------------+------------------+--------------------
   template1           |         79208460 | .0737686268985271
   template0           |         79208460 | .0737686268985271
   postgres            |       1398554636 |  1.30250550433993
   mysql_compatible_db |         79208460 | .0737686268985271
  (4 rows)
  ```

- Query the tablespace size.

  ```
  gaussdb=#  SELECT pg_size_pretty(pg_tablespace_size('pg_default'));
   pg_size_pretty
  ----------------
   345 MB
  (1 row)
  ```

- Query the table size.

  ```
  gaussdb=#  SELECT pg_size_pretty(pg_relation_size('table_name'));
  pg_size_pretty
  ----------------
   0 bytes
  (1 row)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.4.15 How Do I Query the Database Running Time?

Answer: To query the database running time, run the following SQL statement:

```
gaussdb=# SELECT pg_postmaster_start_time() startup_time, (sysdate - startup_time) run_time;
```

9.4.16 How Do I Query the Last Restart Time of a Database?

Answer: To query the latest restart time of the database, run the following SQL statement:

```
gaussdb=# SELECT pg_postmaster_start_time();
   pg_postmaster_start_time
-------------------------------
 2024-08-01 15:34:51.738604+08
(1 row)
```

9.4.17 How Do I Create a Column with the Current Time As the Default Value?

Answer: When creating a table, set the default value of the column to **CURRENT_TIMESTAMP**.

```
gaussdb=# CREATE TABLE tbl (id int, modtime date DEFAULT CURRENT_TIMESTAMP);
```

9.4.18 How Do I Create a Read-only View and Specify Whether the View Is Read-only?

Answer: Create a read-only view.

```
-- Create the test_tb table.
gaussdb=# CREATE TABLE test_tb(c1 int, c2 int);

-- Create the test_v read-only view.
gaussdb=# CREATE  VIEW test_v AS SELECT * FROM test_tb WITH READ ONLY;
```

Check whether the view is read-only.

```
-- Write data to the view to check whether the view is read-only.
gaussdb=# INSERT INTO test_v VALUES (1, 1);
ERROR:  Cannot perform a DML operation on a read-only view.

-- Delete the table and the view.
gaussdb=# DROP VIEW test_v;
gaussdb=# DROP TABLE test_tb;
```

9.4.19 How Do I Disable the System Catalog Structure Modification?

Answer: **allow_system_table_mods** allows system catalog structure modification. This parameter collaborates with GaussDB to help recover a severely damaged database in some cases. You are advised to set **allow_system_table_mods** to **off** and restart the database after setting the parameter.

The configuration and verification methods are as follows:

```
-- Configuration method.
gs_guc set -Z coordinator -Z datanode -N all -I all -c "allow_system_table_mods=off"

-- Verification method
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# SELECT name,setting FROM pg_settings WHERE name = 'allow_system_table_mods';
          name           | setting
-------------------------+---------
 allow_system_table_mods | off
(1 row)
```

9.4.20 How Do I Determine Whether a Column is NULL?

Answer: Use IS NULL and IS NOT NULL to determine whether a column is **NULL**. For example:

```
gaussdb=# SELECT * FROM tab WHERE col1 IS NULL;
```

9.4.21 How Do I Delete Duplicate Rows?

Answer: You can use either of the following methods:

- Method 1:

  This method applies only to distributed tables, and the join condition must contain the distribution key. For replication tables, the ctid of the same data may be different on different DNs. Therefore, this method cannot be used to delete duplicate rows.

  ```
  -- Create a table and insert data into the table.
  gaussdb=# CREATE TABLE tbl_del(c1 varchar,c2 int) distribute by hash(c1);
  gaussdb=# INSERT INTO tbl_del VALUES ('a',1),('a',1);
  gaussdb=# INSERT INTO tbl_del VALUES ('b',2),('b',2);
  gaussdb=# INSERT INTO tbl_del VALUES ('c',3);

  -- Query the deduplicated data.
  gaussdb=# SELECT distinct * FROM tbl_del;
   c1 | c2
  ----+----
   b  |  2
   a  |  1
   c  |  3
  (3 rows)

  -- Run the following SQL statements to delete duplicate data:
  gaussdb=# DELETE FROM tbl_del t1 WHERE ctid <>(
      SELECT max(ctid) FROM tbl_del t2 WHERE t1.c1 = t2.c1
  );

  -- Query data.
  gaussdb=# SELECT * FROM tbl_del;
   c1 | c2
  ----+----
   a  |  1
   b  |  2
   c  |  3
  (3 rows)

  -- Drop.
  gaussdb=# DROP TABLE tbl_del;
  ```

- Method 2:

  ```
  -- Create a table and insert data into the table.
  gaussdb=# CREATE TABLE tbl_del(c1 varchar,c2 int);
  gaussdb=# INSERT INTO tbl_del VALUES ('a',1),('a',1);
  gaussdb=# INSERT INTO tbl_del VALUES ('b',2),('b',2);
  gaussdb=# INSERT INTO tbl_del VALUES ('c',3);

  -- Query the deduplicated data.
  gaussdb=# SELECT distinct * FROM tbl_del;
   c1 | c2
  ----+----
   b  |  2
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 a | 1
 c | 3
(3 rows)

-- Copy the tbl_del2 table.
gaussdb=# CREATE TABLE tbl_del2 AS SELECT * FROM tbl_del;

-- Clear the tbl_del table and insert the deduplicated data into the table.
gaussdb=# TRUNCATE TABLE tbl_del;
gaussdb=# INSERT INTO tbl_del SELECT distinct * FROM tbl_del2;

-- Query.
gaussdb=# SELECT * FROM tbl_del;
 c1 | c2
----+----
 b | 2
 a | 1
 c | 3
(3 rows)

-- Drop.
gaussdb=# DROP TABLE tbl_del,tbl_del2;
```

9.4.22 How Do I Use SQL Statements to View the Database Mode?

Answer: You can use the gs_deployment() system function to view the deployment mode of the current database. The following is an example:

```
gaussdb=#  SELECT gs_deployment();
gs_deployment
---------------
Distribute
(1 row)
```

9.4.23 How Do I Query the Definitions of Tables, Indexes, and Views Using SQL Statements?

- Query the table definition.
  ```
  gaussdb=#  SELECT * FROM pg_get_tabledef('table_name');
  ```
- Query the index definition.
  ```
  gaussdb=#  SELECT oid FROM pg_class WHERE relname='index_name';
  gaussdb=#  SELECT * FROM pg_get_indexdef(oid);
  ```
- Query the view definition.
  ```
  gaussdb=#  SELECT viewname, definition FROM pg_views WHERE viewname='view_name';
  ```

9.4.24 What Are the Advantages of Data Partitioning?

Answer: The advantages are as follows:

- Improved manageability: Tables and indexes are divided into smaller and more manageable units. Maintenance can be performed on a specific part of a table.
- Improved deletion performance: You can delete an entire partition rather than delete data row by row. You can use the ALTER TABLE... DROP PARTITION syntax to delete a partitioned table.
- Improved query performance: You can restrict the volume of data to be checked or manipulated to make queries quicker.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Partition pruning (also known as partition pruning): GaussDB filters out unexpected partitions and scans only the remaining partitions. Partition pruning greatly improves query performance.
- Intelligent partition join: Partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are joint and at least one of these tables is partitioned on the join key. Partition-wise joins break a large join into smaller joins of "identical" data sets. "Identical" here is defined as covering the same set of partitioning key values on both sides of the join, ensuring that only a join of these 'identical' data sets will produce a result and that other data sets do not have to be considered. Currently, list partitioning and hash partitioning are not supported.

9.4.25 What Are the Differences Between UNIQUE Constraints and Unique Indexes?

Answer: The differences between UNIQUE constraints and unique indexes are as follows:

- The concepts of a UNIQUE constraint and a unique index are different.

  A UNIQUE constraint specifies that the values in a column or a group of columns are all unique. A unique index is used to ensure the uniqueness of a column value or the value combination of multiple columns. **CREATE UNIQUE INDEX** creates a unique index.
- The functions of a UNIQUE constraint and a unique index are different.

  Constraints are used to ensure data integrity, and indexes are used to facilitate query.
- The usages of a UNIQUE constraint and a unique index are different.
  - When a UNIQUE constraint is created, a unique index with the same name is automatically created. The index cannot be deleted separately. When the constraint is deleted, the index is automatically deleted. A UNIQUE constraint uses a unique index to ensure data uniqueness.
  - A created unique index is independent and can be deleted separately.
  - If you need both a UNIQUE constraint and a unique index on the same column and want to manage them separately, create a unique index first and then a UNIQUE constraint with the same name.
  - If a field in a table is to be used as a foreign key of another table, the field must have a UNIQUE constraint (or it is a primary key). If the field has only a unique index, an error is reported.

9.4.26 Why Cannot the Table Name or Column Name Be Identified During Query?

Answer: The most common cause is that the table name or column name is enclosed in double quotation marks (") during the creation. When double quotation marks are used, table names and column names (called identifiers) are case-sensitive in storage. Therefore, double quotation marks must be used during query and the case must match that during creation. Some APIs or applications automatically introduce double quotation marks when they are created. Therefore,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

you are advised to use lowercase letters when creating APIs or applications. For example:

```
-- Create a table.
gaussdb=# CREATE TABLE "TB_test" ("Tid" int, num int);

-- If double quotation marks are not added when a table is queried, lowercase letters are used by default. As a result, an error is reported.
gaussdb=# SELECT * FROM TB_test;
ERROR:  relation "tb_test" does not exist
LINE 1: SELECT * FROM TB_test;

gaussdb=# SELECT Tid FROM "TB_test";
ERROR:  Column "tid" does not exist
LINE 1: SELECT Tid FROM "TB_test"

-- Add double quotation marks during query.
gaussdb=# SELECT "Tid"  FROM  "TB_test";
 Tid
-----
(0 rows)

-- Drop the table.
gaussdb=# DROP TABLE "TB_test";
```

9.4.27 What Is the Schema Starting with pg_temp or pg_toast_temp?

Answer: When you query the schemas, the query result contains schemas starting with pg_temp or pg_toast_temp, as shown in the following information.

```
gaussdb=# SELECT nspname FROM pg_namespace WHERE nspname LIKE 'pg_t%' AND nspname != 'pg_toast';
            nspname
--------------------------------
 pg_temp_cn_5001_3_2_3795
 pg_toast_temp_cn_5001_3_2_3795
(2 rows)
```

Tables starting with pg_temp or pg_toast_temp are created when temporary tables are created. The schema starting with pg_temp in each session is visible only to the current session. Therefore, you are advised not to manually delete schemas starting with pg_temp or pg_toast_temp during routine operations.

Temporary tables are visible only in the current session and are automatically deleted after the session ends. The corresponding schemas are also deleted.

9.4.28 After the String Processing Function Is Used, the Query Does Not Use the Index. What Should I Do?

Answer: After a function is used to process an index column, the standard index is not used. You can create an expression index. The following uses the lpad() function as an example.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test (c1 varchar);
gaussdb=# INSERT INTO tbl_test VALUES (generate_series(1,10000));

-- The lpad() function is used for an ordinary index, and the index is not used for query.
gaussdb=# CREATE INDEX idx_test_c1 ON tbl_test(c1);
gaussdb=# SET max_datanode_for_plan = 1;
gaussdb=# SET explain_perf_mode = normal;
-- Part of the execution plan is as follows:
gaussdb=# EXPLAIN SELECT * FROM tbl_test WHERE lpad(c1,5,'0') = '00324';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                                QUERY PLAN
-------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT c1 FROM public.tbl_test WHERE lpad(c1::text, 5, '0'::text) = '00324'::text
 Datanode Name: datanode1
   Seq Scan on tbl_test  (cost=0.00..96.22 rows=25 width=32)
     Filter: (lpad((c1)::text, 5, '0'::text) = '00324'::text)

(8 rows)

-- After the index is changed to an expression index, the index is used for query.
gaussdb=# DROP INDEX idx_test_c1;
gaussdb=# CREATE INDEX idx_test_c1 ON tbl_test(lpad(c1,5,'0'));
gaussdb=# EXPLAIN SELECT * FROM tbl_test WHERE lpad(c1,5,'0') = '00324';
                                QUERY PLAN
-------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes

 Remote SQL: SELECT c1 FROM public.tbl_test WHERE lpad(c1::text, 5, '0'::text) = '00324'::text
 Datanode Name: datanode1
   Bitmap Heap Scan on tbl_test  (cost=1.55..18.09 rows=25 width=32)
     Recheck Cond: (lpad((c1)::text, 5, '0'::text) = '00324'::text)
     ->  Bitmap Index Scan on idx_test_c1  (cost=0.00..1.54 rows=25 width=0)
           Index Cond: (lpad((c1)::text, 5, '0'::text) = '00324'::text)

(10 rows)
-- Drop.
gaussdb=# DROP INDEX idx_test_c1;
gaussdb=# DROP TABLE tbl_test;
```

9.5 Development Design

9.5.1 What Is the Maximum Number of Columns in a Single GaussDB Table?

Answer: 1,600. The specific value varies according to the column type. The column type is not verified during table creation but is verified during data storage. For example, for a column of the bigint type, each column stores 8-byte data. If there are 1,600 columns, 12,800 bytes need to be stored, which exceeds the page capacity (8 KB), and an error is reported during insertion.

NOTE

If the data length of a column exceeds the upper limit, the column may be stored in a TOAST table. The logic is as follows:

- If the total length of a tuple exceeds 2 KB, the longest column is compressed first.
- A column is stored in a TOAST table only when its data length exceeds the size limit after compression and is larger than the TOAST metadata overhead (24 bytes).
- A TOAST table stores the full data of each column, while the main table stores only query metadata.
- After compression, if the total length of a single row exceeds the size of a single page (8 KB), an error is reported regardless of whether a TOAST table is used. If a TOAST table is used, the main table stores its metadata; otherwise, it stores the compressed column data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.5.2 What Are the Types of SQL Languages?

Answer: SQL languages are classified into the following types:

- Data definition language (DDL) is used to define or modify an object in a database, such as a table, an index, or a view.
- Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.
- Data control language (DCL) is used to set or change the permissions of database users or roles.

9.5.3 What Is the Functionality of a Trigger?

Answer: A trigger is a special stored procedure that is triggered and executed by events.

- Constraints can be strengthened to maintain data integrity and consistency.
- Operations in the database can be traced. Unauthorized updates and changes are not allowed.
- Concatenated operations can be performed.

9.5.4 What Are the Differences Between Functions and Stored Procedures?

Answer: Functions and stored procedures are two common database objects that encapsulate a series of SQL statements to complete specific operations. For details about the differences, see **Table 9-2**.

Table 9-2 Differences between functions and stored procedures

Function	Stored Procedure
The identifier for a function is **FUNCTION**.	The identifier for a stored procedure is **PROCEDURE**.
A function must return a value.	A stored procedure may or may not return a value.
Functions can accept input and output parameters.	Stored procedures can accept input and output parameters.
Functions can only call SELECT statements and cannot modify data in tables.	Stored procedures can modify data in tables through statements such as UPDATE, DELETE, and INSERT.
Stored procedures cannot be called within functions.	Both functions and stored procedures can be called within stored procedures, which helps simplify complex operations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	Stored Procedure
Functions are suitable for scenarios that require a single return value, such as performing data calculations, processing character strings, or returning tables.	Stored procedures are suitable for executing DML operations, for example, batch insertion, update, and deletion.

9.5.5 How Do I Query the Triggers in the Current Database and Their Definitions?

Answer: You can use the following method to query the trigger information in the current database:

- Pre-operation: Create a table and a trigger.

```
-- Create a table.
gaussdb=# CREATE TABLE test(c1 int);

-- Create a trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION func_test() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
    UPDATE test SET c1 = c1+200 WHERE id = NEW.id;
    RETURN NEW;
END$$ LANGUAGE plpgsql;

-- Create a trigger.
gaussdb=# CREATE TRIGGER trg_ins_test
AFTER INSERT ON test
FOR EACH ROW
EXECUTE PROCEDURE func_test();
```

- Query a trigger and its definitions in the current database.

```
gaussdb=# SELECT tgname FROM pg_trigger;
    tgname
--------------
 trg_ins_test
(1 row)

gaussdb=# SELECT pg_get_triggerdef(oid) FROM pg_trigger where tgname = 'trg_ins_test';
                                pg_get_triggerdef
-----------------------------------------------------------------------------------------------
 CREATE TRIGGER trg_ins_test AFTER INSERT ON test FOR EACH ROW EXECUTE PROCEDURE
func_test()
(1 row)

-- Query the definition of a trigger function.
gaussdb=# \sf func_test()
CREATE OR REPLACE FUNCTION public.func_test()
 RETURNS trigger
 LANGUAGE plpgsql
 NOT FENCED NOT SHIPPABLE
AS $function$
DECLARE
BEGIN
    UPDATE test SET c1 = c1+200 WHERE id = NEW.id;
    RETURN NEW;END$function$;

-- Drop the table.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- After a table is dropped, the trigger is dropped, but the functions related to the trigger are not dropped.
gaussdb=# DROP TABLE test;
gaussdb=# DROP FUNCTION func_test();
```

9.5.6 How Do I Query Partition Information and Index Information of a Partitioned Table?

Answer: You can use either of the following methods:

- You can use the pg_get_tabledef() function to view the definition of a table. The returned information includes table creation SQL statements, comments, indexes, and constraints.

  Example:

  ```
  gaussdb=# SELECT pg_get_tabledef('table_name');
  ```

- You can query the pg_partition view for partition information and the pg_indexes view for index information.

9.5.7 How Do I Query All Triggers Involved in a Table?

Answer: Run the following SQL statement for queries (change the table name according to the actual situation):

```
-- Query all triggers involved in a table.
gaussdb=# SELECT t1.relname table_name, t2.tgname trigger_name
FROM pg_class t1, pg_trigger t2
WHERE t1.oid = t2.tgrelid AND
    t1.relname = 'tablename';
```

9.5.8 Is There a Limit on the Length of a Database Object Name?

Answer: When a database object is created, its name cannot exceed 63 bytes. If the name exceeds 63 bytes, the database truncates the last byte (not the last character). As a result, half a character may appear.

9.5.9 Which of the Following Factors Should Be Considered When You Select a Database Character Set?

Answer: When selecting the database character set, consider the following factors:

- Requirements for languages supported by applications
- Compatibility of existing data
- Performance requirements
- Character operation requirements
- Cross-platform compatibility
- Storage space and performance requirements
- Nationalization support

9.6 Others

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.1 How Do I Check the Disk Type?

Answer: Check the value of **/sys/block/***<Disk name>***/queue/rotational**. **1** indicates that the disk is an HDD. **0** indicates that the disk is an SSD.

```
cat /sys/block/sdb/queue/rotational
0    --- Solid state disk (SSD).

cat /sys/block/sdb/queue/rotational
1    --- Hard disk drive (HDD).
```

9.6.2 How Do I Query the Active Tuples, Dirty Tuples, and Dirty Tuple Ratios?

Answer: The query method is as follows:

```
-- Query the top 10 active tuples, dirty tuples, and dirty tuple ratios.
gaussdb=#  SELECT schemaname||'.'||relname AS
table_name,pg_size_pretty(pg_relation_size(schemaname||'.'||relname)) AS
table_size,n_dead_tup,n_live_tup,round(n_dead_tup * 100 / (n_live_tup + n_dead_tup),2) AS dead_tup_ratio
FROM pg_stat_all_tables WHERE n_dead_tup >= 1000 ORDER BY dead_tup_ratio DESC LIMIT 10;
                table_name                 | table_size |  n_dead_tup  | n_live_tup | dead_tup_ratio
-------------------------------------------+------------+--------------+------------+----------------
 pg_catalog.pg_statistic                   | 1136 kB    |       1425 |        647 |          68.77
 snapshot.snap_global_stat_all_indexes     | 8072 kB    |       5559 |      16596 |          25.09
 snapshot.snap_summary_stat_all_indexes    | 7672 kB    |       5466 |      16596 |          24.78
 snapshot.snap_class_vital_info            | 3832 kB    |       3171 |      11094 |          22.23
 snapshot.snap_global_file_iostat          | 11 MB      |      11333 |      46687 |          19.53
 snapshot.snap_summary_file_iostat         | 10 MB      |      11325 |      46687 |          19.52
 snapshot.snap_summary_statement           | 13 MB      |       3832 |      16131 |          19.20
 snapshot.snap_summary_statio_all_indexes | 8128 kB    |       4360 |      19952 |          17.93
 snapshot.snap_global_statio_all_indexes  | 8456 kB    |       4275 |      19952 |          17.65
 snapshot.snap_global_stat_all_tables      | 8216 kB    |       3038 |      16409 |          15.62
(10 rows)
```

NOTE

Dirty tuples can be used as an indicator to determine the table bloat degree. A high ratio of dirty tuples may cause table bloat. Therefore, you need to reclaim the space in a timely manner to prevent database performance from being affected. For details, see "SQL Reference > SQL Syntax > V > VACUUM" in *Reference*.

9.6.3 What Are the Main Differences Between VACUUM and VACUUM FULL?

Answer: In GaussDB, both VACUUM and VACUUM FULL are used to clear tablespaces. However, they have obvious differences, as shown in the following:

- VACUUM: traverses all rows in the table and marks the dead tuples rows (rows that are no longer referenced by any active transaction) in the table as deleted. The dead tuple rows are not physically deleted, and the space of the dead tuple rows is marked as reusable. During VACUUM, the table is not locked, and other transactions can access the data table normally.
- VACUUM FULL: physically deletes dead tuples rows from a table to release space for the OS. During VACUUM FULL, the table is locked and cannot be accessed by other transactions.

VACUUM is a lightweight command that does not block database operations and has little impact on database performance. VACUUM FULL is a heavyweight command that blocks database operations and has great impact on database

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

performance. In practice, select a value based on the database load and service requirements.

9.6.4 What Are the Differences Between pg_terminate_backend and pg_terminate_session in GaussDB?

Answer: In GaussDB, **pg_terminate_backend** and **pg_terminate_session** are used to terminate database sessions.

- pg_terminate_backend is used to terminate a backend thread. It forcibly terminates any operation that is being performed by the thread, including ongoing transactions. It can be used to terminate a long-running query or a blocked transaction.
- pg_terminate_session is used to terminate a backend session. It terminates all operations being performed in the session, but does not forcibly terminate ongoing transactions. It can be used to terminate the session of a user without affecting the sessions of other users.

9.6.5 How Do I Display Database File Information?

Answer: You can use the gs_controldata tool to display information about the **pg_control** file in the database data directory.

```
-- View the version of the gs_controldata tool.
[omm1@node41 ~]$ gs_controldata -V
gs_controldata (GaussDB Kernel 505.2.RC1 build 939a176f) compiled at 2024-04-30 20:05:50 last mr 17511 release

-- Locate the path of the pg_control file.
[root@node41 ~]# find / -name pg_control
/opt/software/db5052/gaussdb/data/slave31/global/pg_control
/opt/software/db5052/gaussdb/data/coordinator/global/pg_control
/opt/software/db5052/gaussdb/data/master1/global/pg_control
/opt/software/db5052/gaussdb/data/slave21/global/pg_control

-- Check the information about the pg_control file. The DATADIR directory must be written to the lower-level directory of the data layer in the queried path. The master1 directory is used as an example.
[omm1@node41 ~]$ gs_controldata /opt/software/db5052/gaussdb/data/master1
pg_control version number:            923
Catalog version number:               201611171
Database system identifier:           18020659789221943256
Database cluster state:               in production
pg_control last modified:             Sat 11 May 2024 10:43:58 AM CST
Latest checkpoint location:           0/134044D8
Prior checkpoint location:            0/13403BB8
Latest checkpoint's REDO location:    0/13404458
Latest checkpoint's TimeLineID:       1
Latest checkpoint's full_page_writes: off
Latest checkpoint's NextXID:          70025
Latest checkpoint's NextOID:          24576
Latest checkpoint's NextMultiXactId:  2
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        3
Latest checkpoint's oldestXID's DB:   2
Latest checkpoint's oldestActiveXID:  70025
Latest checkpoint's remove lsn:       0/1
Time of latest checkpoint:            Sat 11 May 2024 10:43:57 AM CST
Minimum recovery ending location:     0/0
Backup start location:                0/0
Backup end location:                  0/0
End-of-backup record required:        no
Current wal_level setting:            hot_standby
Current max_connections setting:      1084
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Current max_prepared_xacts setting:   800
Current max_locks_per_xact setting:   256
Maximum data alignment:               8
Database block size:                  8192
Blocks per segment of large relation: 131072
WAL block size:                       8192
Bytes per WAL segment:                16777216
Maximum length of identifiers:        64
Maximum columns in an index:          32
Maximum size of a TOAST chunk:        1996
Date/time type storage:               64-bit integers
Float4 argument passing:              by value
Float8 argument passing:              by value
Database system TimeLine:             1
```

9.6.6 How Often Are Xlogs in GaussDB Automatically Recycled?

Answer: The GaussDB background thread for automatically recycling Xlogs checks Xlogs every minute and recycles the Xlogs that meet the conditions.

Log recycling is triggered only when the size of logs reaches a certain value. The formula for calculating the value is as follows: Value of **wal_keep_segments** + Value of **checkpoint_segments** x 2 + 3. If **checkpoint_segments** is set to **128** and **wal_keep_segments** is set to **128**, the size of logs is (128 + 128 x 2 + 3) x 16 MB.

9.6.7 What Is a Dead Tuple?

Answer: Currently, GaussDB uses the Astore. Data update is implemented in append mode. That is, data is only marked as "deleted" after an UPDATE or DELETE operation, and the space is not released immediately. This processing is necessary for multi-version concurrency control (MVCC). If a tuple version may still be viewed by other transactions, the tuple cannot be deleted.

After a transaction is committed, the expired tuple version is no longer valid for the transaction. These tuples are marked as dead tuples.

Dead tuples are generated because GaussDB does not have an independent rollback segment and records historical versions on data pages. Excessive dead tuples occupy a large amount of storage space. If a large number of dead tuples are not cleared, indexes will point to a large number of invalid pages, affecting the efficiency of SQL statement scan through indexes.

9.6.8 How Do I Query Dead Tuples of a Table?

Answer: Query pg_stat_all_tables about dead tuples.

```
gaussdb=# SELECT relname,n_live_tup+n_dead_tup all_tup,n_dead_tup,last_autovacuum,last_autoanalyze
FROM pg_stat_all_tables WHERE relname='test02';
relname | all_tup   | n_dead_tup |          last_autovacuum   |      last_autoanalyze
---------+---------+------------+-------------------------------+-------------------------------
test02  |  500000 |      10000 | 2023-04-26 14:09:33.933982+08 | 2023-04-26 14:17:34.0364+08
```

9.6.9 What Is OID?

Answer: Object identifier (OID) is the unique identifier of a database object, which can be a database, table, index, view, or so on.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.10 What Is UDF?

Answer: Abbreviation of user-defined function (UDF). It indicates that users can define a function and embed it in GaussDB to implement specific functionalities in different service scenarios.

9.6.11 What Is the Depth of Nested Calling of a Stored Procedure?

Answer: There is no limit on the depth of calling a stored procedure, but the depth cannot exceed the maximum stack depth. The maximum stack depth is controlled by the GUC parameter **max_stack_depth**, which limits the maximum depth of nested calling of a stored procedure in a session.

9.6.12 What Is TOAST?

Answer: The oversized-attribute storage technique (TOAST) is a special table in GaussDB and is used to store ultra-long data fields. When the length of the complete structure (including columns) of a tuple exceeds the size limit of a page, GaussDB enables the TOAST mechanism to compress or slice these columns into multiple physical rows and store them to another table (TOAST table).

9.6.13 What Is the Function of the Sandbox?

Answer: A sandbox is a security mechanism. It provides an isolated and restricted operating environment for untrusted programs so that high-risk programs can access only the resources that they need to access, reduce the risks of high-risk program vulnerabilities, such as system damage, other process damage, and access to confidential and private data.

9.6.14 What Is a Vector Database?

Answer: A vector database is a database system designed for storing, managing, and retrieving high-dimensional vector data. Compared with traditional relational databases, they differ significantly in data model, indexing mechanism, and query processing. Vector databases use optimized indexing technologies to improve the retrieval efficiency of unstructured data. In addition, they can support complex data analysis and machine learning models, making data processing more efficient and intelligent.

Vector databases are mainly used in the following scenarios:

- Recommendation systems in scenarios such as video recommendation, news recommendation, and shopping recommendation.
- Image recognition in scenarios such as product search by image and facial recognition.
- Unstructured data processing in scenarios such as audio, video, and image data conversion (to vector data) for criticality analysis.
- LLM-based intelligent question answering systems, for example, vector databases are used to improve semantic search engines, and provide more accurate search results by storing and retrieving vector representations of text contents.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.15 What Is a GIN Index?

Answer: GIN stands for generalized inverted index (GIN), which is designed for cases where the items to be indexed are composite values, and queries handled by the index need to search for element values within the composite items. For example, search for a specific word in a document, which is a collection of words. The GIN greatly accelerates fuzzy query.

The GIN meets the following conditions:

- Multiple GINs are created on the same column of the same table.
- GINs use different parsers (that is, different delimiters).
- The column is used in a query, and an index scan is used in the execution plan.
- To avoid different query results caused by different GINs, ensure that only one GIN is available on a column of a physical table.

9.6.16 Can a Stored Procedure Call a Trigger?

Answer: Yes. A stored procedure can trigger a trigger by executing an SQL statement (such as UPDATE).

9.6.17 How Do I Query Several Random Rows in a Table?

Answer: You can use the following method to query several random data records in a table:

```
-- Create a table and insert 100 data records into the table.
gaussdb=# CREATE TABLE test(c1 int);
gaussdb=# INSERT INTO test VALUES (generate_series(1,100));

-- Query four random rows of data in the table.
gaussdb=# SELECT * FROM test ORDER BY random() limit 4;
 c1
----
 99
 50
 78
 35
(4 rows)
```

9.6.18 Why Is There a Gap in the Sequence Number?

Answer: To improve concurrency, each session caches a certain number of sequence values in the private memory space of the user based on the value of the **CACHE** parameter of the sequence. After the session ends, the sequence value cached in the session is discarded. Therefore, the sequence value jumps.

```
-- Create a table and a sequence.
gaussdb=# CREATE SEQUENCE seq_test CACHE 10;

-- Use nextval to view the value of the next sequence and exit the session.
gaussdb=# SELECT nextval('seq_test');
 nextval
---------
       1
(1 row)
gaussdb=# SELECT nextval('seq_test');
 nextval
---------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      2
(1 row)

-- Use nextval() in another session to view the value of the next sequence.
gaussdb=# SELECT nextval('seq_test');
 nextval
---------
      11
(1 row)

-- Drop the sequence.
gaussdb=# DROP SEQUENCE seq_test;
```

9.6.19 What Do I Do If the Result of the power() Function Is Too Large?

Answer: Generally, the calculation result of the power() function is inaccurate. For example, the accurate value of 2 to the power of 60 is 1,152,921,504,606,846,976, and the value of **SELECT power (2,60)** is 1.15292150460685e+18, which have errors on the accuracy. The solution is as follows:

```
gaussdb=# SELECT power(2,60);
        power
----------------------
 1.15292150460685e+18
(1 row)

-- The following query result indicates that the return value of two input parameters of the double
precision type is also of the double precision type, which is accurate to 16 decimal places.
gaussdb=# \df power;
                                                 List of functions
   Schema   | Name  | Result data type |        Argument data types         |  Type  | fencedmode | propackage |
prokind
------------+-------+------------------+------------------------------------+--------+------------+------------+---------
 pg_catalog | power | double precision | double precision, double precision | normal | f          | f          | f
 pg_catalog | power | numeric          | numeric, numeric                   | normal | f          | f          | f
(2 rows)

-- Perform the following operations to query:
gaussdb=# SELECT power(2::numeric,60::numeric);
                power
--------------------------------------
 1152921504606846976.0000000000000000
(1 row)

-- Use the round() function to cancel the decimal point.
gaussdb=# SELECT round(power(2::numeric,60::numeric),0);
        round
---------------------
 1152921504606846976
(1 row)
```

9.6.20 What Is CTID?

Answer: CTID identifies specific physical rows by blocks and locations in the table.

- The format is (*blockid*,*itemid*), for example, **(0,1)**, in which **0** indicates the block ID, and **(0,1)** indicating the first record in block 0.
- Indexes use CTID to point to physical rows.
- The CTID of a row changes when the row is updated. Therefore, the CTID cannot be used as a long-term identifier.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.21 How Do I Query the *M*th Largest Data in a Column of a Table?

Answer: You can use the window function row_number(), rank(), or dense_rank() according to the actual situation.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(name varchar(20),tamo int);
gaussdb=# INSERT INTO tbl_test VALUES('Bob',7000),('Jack',6500),('July',6800);
gaussdb=# INSERT INTO tbl_test VALUES('Luna',7200),('Henry',7000),('John',8000);

-- Run the following SQL statement to query the data whose transaction amount (tamo) ranks third:
gaussdb=# SELECT * FROM (SELECT *,dense_rank() over(order by tamo desc) rank FROM tbl_test) WHERE
rank = 3;
 name  | tamo | rank
-------+------+------
 Bob   | 7000 |    3
 Henry | 7000 |    3
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
```

9.6.22 How Do I Query Data Whose Columns Contain Non-digit Characters?

Answer: You can run the following SQL statements to query the information:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(c1 varchar);
gaussdb=# INSERT INTO tbl_test VALUES('1'),('11'),('1a3'),('1b'),('d1');

-- Query data.
gaussdb=# SELECT * FROM tbl_test WHERE NOT regexp_like(c1,'^[[:digit:]]*$');
 c1
-----
 1a3
 1b
 d1
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
```

9.6.23 How Do I Change the Column Sequence?

Answer: You can use the following method to change the column sequence:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(
    id varchar(5),
    age int CHECK(age>0 and age<=200),
    name varchar(20),
    CONSTRAINT pk_test_id PRIMARY KEY (id)
);
gaussdb=# INSERT INTO tbl_test VALUES ('00010',18,'Bob');
gaussdb=# INSERT INTO tbl_test VALUES ('00011',26,'Jack');
gaussdb=# INSERT INTO tbl_test VALUES ('00012',23,'Jim');

-- View the table definition.
gaussdb=# \d tbl_test

-- Create the tbl_test_new table and change the sequence of the age and name columns.
gaussdb=# CREATE TABLE tbl_test_new(id varchar(5),name varchar(20),age int);
gaussdb=# ALTER TABLE tbl_test_new ADD CONSTRAINT pk_test_new_id PRIMARY KEY (id);
gaussdb=# ALTER TABLE tbl_test_new ADD CONSTRAINT chk_test_new_age CHECK(age>10 AND
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
age<=200);

-- Insert data into the new table.
gaussdb=# INSERT INTO tbl_test_new SELECT id,name,age FROM tbl_test;

-- Change the table names.
gaussdb=# ALTER TABLE tbl_test RENAME TO tbl_test_bak;
gaussdb=# ALTER TABLE tbl_test_new RENAME TO tbl_test;

-- Drop.
gaussdb=# DROP TABLE tbl_test;
gaussdb=# DROP TABLE tbl_test_bak;
```

9.6.24 What Is the Difference Between count(*) and count(column)?

Answer: The count() function is an aggregate function used to calculate the number of rows. The differences are as follows:

- count(*) collects statistics on the number of rows in all columns. Data whose value is **NULL** is not ignored during statistics collection.
- count(column) contains only the column name column. Data whose value is **NULL** is ignored during statistics collection.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test(c1 int);
gaussdb=# INSERT INTO test VALUES(1),(2),(3);
gaussdb=# INSERT INTO test VALUES(NULL);

--count(*) counts the number of rows and does not ignore the data whose value is NULL.
gaussdb=# SELECT count(*) FROM test;
 count
-------
     4
(1 row)

-- count(column) counts the number of rows and ignores the data whose value is NULL.
gaussdb=# SELECT count(c1) from test;
 count
-------
     3
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE test;
```

9.6.25 What Wildcards Are Supported in GaussDB? How Do I Use Them?

Answer: GaussDB supports the following three types of wildcards:

- %: indicates any number of characters, including 0. It is used in the LIKE and NOT LIKE statements.
- _: indicates a character, which is used in LIKE and NOT LIKE statements.
- *: indicates any number of characters, including 0. It is used in some meta-commands.

  Example:

  ```
  -- Use wildcards to indicate any number of characters and query the tbl_test table for the data that starts with col1 and ends with ab.
  gaussdb=# SELECT * FROM tbl_test WHERE col1 LIKE 'ab%';
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Use a wildcard to query the data of any single character string starting with a and ending with b in the col1 column of the tbl_test table.
gaussdb=# SELECT * FROM tbl_test WHERE col1 LIKE 'a_b';

-- Query all tables whose names start with tbl.
gaussdb=# \dt tbl*
```

9.6.26 What Are the Differences Between LIMIT 2, LIMIT 2,3, and LIMIT 2 OFFSET 3?

Answer: LIMIT and OFFSET are keywords used to limit the number of query results returned in the SELECT statement.

- **LIMIT 2**: Only the first two rows of data are returned.
- **LIMIT 2,3**: A total of three rows of data from the third row are returned.
- **LIMIT 2 OFFSET 3**: A total of two rows of data from the fourth row are returned.

9.6.27 What Is the Relationship Between an Empty String and NULL?

Answer: Whether an empty string in GaussDB is NULL depends on the compatibility mode of the database.

- A null string is judged as NULL in ORA-compatible mode.

```
gaussdb=# CREATE DATABASE db_test1 DBCOMPATIBILITY = 'ORA';
gaussdb=# \c db_test1
db_test1=# SELECT '' IS NULL;
 ?column?
----------
 t
(1 row)
```

- A null string is not judged as NULL in MySQL-compatible mode.

```
gaussdb=# CREATE DATABASE db_test2 DBCOMPATIBILITY = 'MYSQL';
gaussdb=# \c db_test2
db_test2=# SELECT '' IS NULL;
 ?column?
----------
 f
(1 row)
```

9.6.28 Which Date/Time Types Contain the Time Zone and Which Do Not?

Answer: The date, time, timestamp, smalldatetime and datea types do not contain the time zone, and the timetz and timestamptz types contain the time zone.

```
-- The timestamp type does not contain the time zone.
gaussdb=# SELECT '2019-01-01'::timestamp;
      timestamp
---------------------
 2019-01-01 00:00:00
(1 row)

-- The timestamptz type contains the time zone.
gaussdb=# SELECT '2019-01-01'::timestamptz;
      timestamptz
------------------------
 2019-01-01 00:00:00+08
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

9.6.29 If No Distribution Key Is Specified, How Will Data Be Stored?

Answer: If no distribution key is specified during table creation, data is stored as follows:

- If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution key is the column corresponding to the primary key or unique constraint.

```
gaussdb=# CREATE TABLE tb_test1
(
    W_WAREHOUSE_SK           INTEGER           PRIMARY KEY,
    W_WAREHOUSE_ID           CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tb_test1_pkey" for table "tb_test1"
CREATE TABLE

gaussdb=# SELECT getdistributekey('tb_test1');
 getdistributekey
------------------
 w_warehouse_sk
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE tb_test1;
```

- If the primary key or unique constraint is not included during table creation but there are columns whose data types can be used as distribution keys, hash distribution is selected. The distribution key is the first column whose data type can be used as a distribution key.

```
gaussdb=# CREATE TABLE tb_test2
(
    W_WAREHOUSE_SK           INTEGER                   ,
    W_WAREHOUSE_ID           CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_sk' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# SELECT getdistributekey('tb_test2');
 getdistributekey
------------------
 w_warehouse_sk
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE tb_test2;
```

- If the primary key or unique constraint is not included during table creation and no column whose data type can be used as a distribution key exists, round-robin distribution is selected.

```
gaussdb=# CREATE TABLE tb_test3
(
    W_WAREHOUSE_ID           CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

gaussdb=# SELECT getdistributekey('tb_test3');
 getdistributekey
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
------------------
 w_warehouse_id
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE tb_test3;
```

9.6.30 What Is Data Skew and How Do I Check Data Skew?

Answer: Data skew indicates that data is unevenly distributed on multiple DNs. For a hash table, an inappropriate distribution key may cause data skew or poor efficiency on certain DNs. Therefore, you need to periodically check or monitor the table to ensure that data is evenly distributed on each DN. Run the following statement to check the number of tuples on each DN:

```
gaussdb=# SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM tablename
GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count DESC;
```

If tuple numbers vary greatly (several times or tenfold) in each DN, a data skew occurs. Change the data distribution key. ALTER TABLE cannot change distribution keys. Therefore, you need to rebuild a table when changing its distribution keys.

9.6.31 How Do I Ignore Case When Using LIKE for Fuzzy Search?

Answer: You can change LIKE to ILIKE or use the upper or lower function. Example:

```
-- Perform pre-operations: creating tables and inserting data.
gaussdb=# CREATE TABLE tbl_test1(c1 varchar);
gaussdb=# INSERT INTO tbl_test1 VALUES ('EEE'),('ABC'),('abc'),('aabccd');

-- When LIKE is used for fuzzy query, all uppercase character strings are filtered out, which does not meet
the expectation.
gaussdb=# SELECT * FROM tbl_test1 WHERE c1 LIKE 'ab%';
 c1
-----
 abc
(1 row)

-- Use ILIKE for fuzzy search.
gaussdb=# SELECT * FROM tbl_test1 WHERE c1 ILIKE 'ab%';
 c1
-----
 ABC
 abc
(2 rows)

-- Use the upper() function.
gaussdb=# SELECT * FROM tbl_test1 WHERE upper(c1) LIKE 'AB%';
 c1
-----
 ABC
 abc
(2 rows)

-- Drop.
gaussdb=# DROP TABLE tbl_test1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.